
Document Number: MCUXSDKAPIRM
Rev 2.11.0
Jan 2022

MCUXpresso SDK API Reference Manual

NXP Semiconductors



Contents

Chapter 1 Introduction

Chapter 2 Trademarks

Chapter 3 Architectural Overview

Chapter 4 ANATOP_AI driver

4.1	Overview	7
4.2	Macro Definition Documentation	13
4.2.1	FSL_ANATOP_AI_DRIVER_VERSION	13
4.2.2	AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS	13
4.2.3	AI_PHY_LDO_CTRL0_OUTPUT_TRG	13
4.2.4	AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD	14
4.2.5	AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP	14
4.2.6	AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF	14
4.2.7	AI_BANDGAP_CTRL0_REFTOP_VBGADJ	14
4.3	Enumeration Type Documentation	14
4.3.1	anatop_ai_itf_t	14
4.3.2	anatop_ai_reg_t	15

Chapter 5 Clock Driver

5.1	Overview	17
5.2	Data Structure Documentation	32
5.2.1	struct clock_group_config_t	32
5.2.2	struct clock_arm_pll_config_t	32
5.2.3	struct clock_usb_pll_config_t	33
5.2.4	struct clock_pll_ss_config_t	33
5.2.5	struct clock_sys_pll2_config_t	34
5.2.6	struct clock_sys_pll1_config_t	34
5.2.7	struct clock_video_pll_config_t	34
5.2.8	struct clock_sys_pll1_gpc_config_t	35
5.2.9	struct clock_enet_pll_config_t	35
5.2.10	struct clock_root_config_t	36

Section No.	Title	Page No.
5.2.11	struct clock_root_setpoint_config_t	36
5.3	Macro Definition Documentation	36
5.3.1	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	36
5.3.2	FSL_CLOCK_DRIVER_VERSION	37
5.3.3	LPADC_CLOCKS	37
5.3.4	ADC_ETC_CLOCKS	37
5.3.5	AOI_CLOCKS	37
5.3.6	DCDC_CLOCKS	37
5.3.7	DCDC_CLOCKS	38
5.3.8	SRC_CLOCKS	38
5.3.9	GPC_CLOCKS	38
5.3.10	SSARC_CLOCKS	38
5.3.11	WDOG_CLOCKS	38
5.3.12	EWM_CLOCKS	39
5.3.13	SEMA_CLOCKS	39
5.3.14	MU_CLOCKS	39
5.3.15	EDMA_CLOCKS	39
5.3.16	FLEXRAM_CLOCKS	39
5.3.17	LMEM_CLOCKS	40
5.3.18	FLEXSPI_CLOCKS	40
5.3.19	RDC_CLOCKS	40
5.3.20	SEMC_CLOCKS	40
5.3.21	XECC_CLOCKS	40
5.3.22	IEE_CLOCKS	41
5.3.23	KEYMANAGER_CLOCKS	41
5.3.24	PUF_CLOCKS	41
5.3.25	OCOTP_CLOCKS	41
5.3.26	CAAM_CLOCKS	41
5.3.27	XBAR_CLOCKS	42
5.3.28	IOMUXC_CLOCKS	42
5.3.29	GPIO_CLOCKS	42
5.3.30	KPP_CLOCKS	42
5.3.31	FLEXIO_CLOCKS	42
5.3.32	DAC_CLOCKS	43
5.3.33	CMP_CLOCKS	43
5.3.34	PIT_CLOCKS	43
5.3.35	GPT_CLOCKS	43
5.3.36	TMR_CLOCKS	43
5.3.37	ENC_CLOCKS	44
5.3.38	PWM_CLOCKS	44
5.3.39	FLEXCAN_CLOCKS	44
5.3.40	LPUART_CLOCKS	44
5.3.41	LPI2C_CLOCKS	45
5.3.42	LPSPI_CLOCKS	45

Section No.	Title	Page No.
5.3.43	EMVSIM_CLOCKS	45
5.3.44	ENET_CLOCKS	45
5.3.45	ENETQOS_CLOCKS	45
5.3.46	USB_CLOCKS	46
5.3.47	CDOG_CLOCKS	46
5.3.48	USDHC_CLOCKS	46
5.3.49	ASRC_CLOCKS	46
5.3.50	MQS_CLOCKS	46
5.3.51	PDM_CLOCKS	47
5.3.52	SPDIF_CLOCKS	47
5.3.53	SAI_CLOCKS	47
5.3.54	PXP_CLOCKS	47
5.3.55	GPU2D_CLOCKS	47
5.3.56	LCDIF_CLOCKS	48
5.3.57	LCDIFV2_CLOCKS	48
5.3.58	MIPI_DSI_HOST_CLOCKS	48
5.3.59	MIPI_CSI2RX_CLOCKS	48
5.3.60	CSI_CLOCKS	48
5.3.61	DCIC_CLOCKS	49
5.3.62	DMAMUX_CLOCKS	49
5.3.63	XBARA_CLOCKS	49
5.3.64	XBARB_CLOCKS	49
5.3.65	CLOCK_GetCoreSysClkFreq	49
5.4	Enumeration Type Documentation	49
5.4.1	clock_lpcg_t	49
5.4.2	clock_name_t	53
5.4.3	clock_root_t	54
5.4.4	clock_root_mux_source_t	55
5.4.5	clock_group_t	71
5.4.6	clock_osc_t	71
5.4.7	clock_gate_value_t	71
5.4.8	clock_mode_t	72
5.4.9	clock_usb_src_t	72
5.4.10	clock_usb_phy_src_t	72
5.4.11	_clock_pll_clk_src	72
5.4.12	clock_pll_post_div_t	72
5.4.13	clock_pll_t	73
5.4.14	clock_pfd_t	73
5.4.15	clock_control_mode_t	73
5.4.16	clock_24MOsc_mode_t	73
5.4.17	clock_16MOsc_source_t	73
5.4.18	clock_1MHzOut_behavior_t	74
5.4.19	clock_level_t	74

Section No.	Title	Page No.
5.5	Function Documentation	74
5.5.1	CLOCK_SetRootClockMux	74
5.5.2	CLOCK_GetRootClockMux	74
5.5.3	CLOCK_GetRootClockSource	75
5.5.4	CLOCK_SetRootClockDiv	75
5.5.5	CLOCK_GetRootClockDiv	75
5.5.6	CLOCK_PowerOffRootClock	76
5.5.7	CLOCK_PowerOnRootClock	77
5.5.8	CLOCK_SetRootClock	77
5.5.9	CLOCK_ControlGate	77
5.5.10	CLOCK_EnableClock	77
5.5.11	CLOCK_DisableClock	78
5.5.12	CLOCK_SetGroupConfig	78
5.5.13	CLOCK_GetFreq	78
5.5.14	CLOCK_GetRootClockFreq	78
5.5.15	CLOCK_GetM7Freq	79
5.5.16	CLOCK_GetM4Freq	79
5.5.17	CLOCK_IsPIIBypassed	79
5.5.18	CLOCK_IsPIIEnabled	79
5.5.19	CLOCK_GetRtcFreq	80
5.5.20	CLOCK_OSC_SetOsc48MControlMode	80
5.5.21	CLOCK_OSC_EnableOsc48M	80
5.5.22	CLOCK_OSC_SetOsc48MDiv2ControlMode	80
5.5.23	CLOCK_OSC_EnableOsc48MDiv2	81
5.5.24	CLOCK_OSC_SetOsc24MControlMode	81
5.5.25	CLOCK_OSC_EnableOsc24M	81
5.5.26	CLOCK_OSC_GateOsc24M	81
5.5.27	CLOCK_OSC_SetOsc24MWorkMode	82
5.5.28	CLOCK_OSC_SetOscRc400MControlMode	82
5.5.29	CLOCK_OSC_EnableOscRc400M	82
5.5.30	CLOCK_OSC_GateOscRc400M	82
5.5.31	CLOCK_OSC_TrimOscRc400M	83
5.5.32	CLOCK_OSC_SetOscRc400MRefClkDiv	84
5.5.33	CLOCK_OSC_SetOscRc400MFastClkCount	84
5.5.34	CLOCK_OSC_SetOscRc400MHysteresisValue	84
5.5.35	CLOCK_OSC_BypassOscRc400MTuneLogic	84
5.5.36	CLOCK_OSC_EnableOscRc400MTuneLogic	85
5.5.37	CLOCK_OSC_FreezeOscRc400MTuneValue	85
5.5.38	CLOCK_OSC_SetOscRc400MTuneValue	85
5.5.39	CLOCK_OSC_Set1MHzOutputBehavior	85
5.5.40	CLOCK_OSC_SetLocked1MHzCount	86
5.5.41	CLOCK_OSC_CheckLocked1MHzErrorFlag	86
5.5.42	CLOCK_OSC_GetCurrentOscRc400MFastClockCount	86
5.5.43	CLOCK_OSC_GetCurrentOscRc400MTuneValue	86
5.5.44	CLOCK_OSC_SetOsc16MControlMode	86

Section No.	Title	Page No.
5.5.45	CLOCK_OSC_SetOsc16MConfig	87
5.5.46	CLOCK_InitArmPll	87
5.5.47	CLOCK_CalcArmPllFreq	87
5.5.48	CLOCK_InitArmPllWithFreq	88
5.5.49	CLOCK_CalcPllSpreadSpectrum	89
5.5.50	CLOCK_InitSysPll1	89
5.5.51	CLOCK_GPC_SetSysPll1OutputFreq	89
5.5.52	CLOCK_InitSysPll2	89
5.5.53	CLOCK_IsSysPll2PfdEnabled	90
5.5.54	CLOCK_InitSysPll3	90
5.5.55	CLOCK_IsSysPll3PfdEnabled	90
5.5.56	CLOCK_SetPllBypass	91
5.5.57	CLOCK_CalcAvPllFreq	92
5.5.58	CLOCK_InitAudioPllWithFreq	92
5.5.59	CLOCK_InitAudioPll	92
5.5.60	CLOCK_GPC_SetAudioPllOutputFreq	93
5.5.61	CLOCK_InitVideoPllWithFreq	93
5.5.62	CLOCK_InitVideoPll	93
5.5.63	CLOCK_GPC_SetVideoPllOutputFreq	93
5.5.64	CLOCK_GetPllFreq	94
5.5.65	CLOCK_InitPfd	94
5.5.66	CLOCK_DeinitPfd	94
5.5.67	CLOCK_GetPfdFreq	95
5.5.68	CLOCK_EnableUsbhs0Clock	96
5.5.69	CLOCK_EnableUsbhs1Clock	96
5.5.70	CLOCK_EnableUsbhs0PhyPllClock	97
5.5.71	CLOCK_DisableUsbhs0PhyPllClock	97
5.5.72	CLOCK_EnableUsbhs1PhyPllClock	97
5.5.73	CLOCK_DisableUsbhs1PhyPllClock	98
5.5.74	CLOCK_OSCPLL_LockControlMode	98
5.5.75	CLOCK_OSCPLL_LockWhiteList	98
5.5.76	CLOCK_OSCPLL_SetWhiteList	98
5.5.77	CLOCK_OSCPLL_IsSetPointImplemented	98
5.5.78	CLOCK_OSCPLL_ControlByUnassignedMode	99
5.5.79	CLOCK_OSCPLL_ControlBySetPointMode	99
5.5.80	CLOCK_OSCPLL_ControlByCpuLowPowerMode	99
5.5.81	CLOCK_OSCPLL_SetCurrentClockLevel	100
5.5.82	CLOCK_OSCPLL_ControlByDomainMode	100
5.5.83	CLOCK_ROOT_LockControlMode	100
5.5.84	CLOCK_ROOT_LockWhiteList	101
5.5.85	CLOCK_ROOT_SetWhiteList	101
5.5.86	CLOCK_ROOT_IsSetPointImplemented	101
5.5.87	CLOCK_ROOT_ControlByUnassignedMode	102
5.5.88	CLOCK_ROOT_ConfigSetPoint	102
5.5.89	CLOCK_ROOT_EnableSetPointControl	102

Section No.	Title	Page No.
5.5.90	CLOCK_ROOT_ControlBySetPointMode	102
5.5.91	CLOCK_ROOT_ControlByDomainMode	103
5.5.92	CLOCK_LPCG_LockControlMode	103
5.5.93	CLOCK_LPCG_LockWhiteList	103
5.5.94	CLOCK_LPCG_SetWhiteList	104
5.5.95	CLOCK_LPCG_IsSetPointImplemented	104
5.5.96	CLOCK_LPCG_ControlByUnassignedMode	104
5.5.97	CLOCK_LPCG_ControlBySetPointMode	105
5.5.98	CLOCK_LPCG_ControlByCpuLowPowerMode	105
5.5.99	CLOCK_LPCG_SetCurrentClockLevel	105
5.5.100	CLOCK_LPCG_ControlByDomainMode	106

Chapter 6 DCDC_SOC driver

6.1	Overview	107
6.2	Data Structure Documentation	115
6.2.1	struct dcde_config_t	115
6.2.2	struct dcde_min_power_config_t	115
6.2.3	struct dcde_detection_config_t	116
6.2.4	struct dcde_loop_control_config_t	116
6.2.5	struct dcde_internal_regulator_config_t	118
6.2.6	struct dcde_low_power_config_t	118
6.2.7	struct dcde_setpoint_config_t	118
6.3	Macro Definition Documentation	119
6.3.1	FSL_DCDC_DRIVER_VERSION	120
6.3.2	STANDBY_MODE_VDD1P0_TARGET_VOLTAGE	120
6.3.3	STANDBY_MODE_VDD1P8_TARGET_VOLTAGE	120
6.3.4	BUCK_MODE_VDD1P0_TARGET_VOLTAGE	120
6.3.5	BUCK_MODE_VDD1P8_TARGET_VOLTAGE	120
6.4	Enumeration Type Documentation	121
6.4.1	_dcde_status_flags	121
6.4.2	_dcde_setpoint_map	121
6.4.3	dcde_control_mode_t	121
6.4.4	dcde_trim_input_mode_t	122
6.4.5	dcde_standby_mode_1P0_target_vol_t	122
6.4.6	dcde_standby_mode_1P8_target_vol_t	123
6.4.7	dcde_buck_mode_1P0_target_vol_t	123
6.4.8	dcde_buck_mode_1P8_target_vol_t	124
6.4.9	dcde_comparator_current_bias_t	125
6.4.10	dcde_peak_current_threshold_t	125
6.4.11	dcde_clock_source_t	126
6.4.12	dcde_voltage_output_sel_t	126

Section No.	Title	Page No.
6.5	Function Documentation	126
6.5.1	DCDC_Init	126
6.5.2	DCDC_Deinit	126
6.5.3	DCDC_GetDefaultConfig	126
6.5.4	DCDC_EnterLowPowerModeViaStandbyRequest	127
6.5.5	DCDC_EnterLowPowerMode	127
6.5.6	DCDC_EnterStandbyMode	127
6.5.7	DCDC_SetVDD1P0StandbyModeTargetVoltage	128
6.5.8	DCDC_GetVDD1P0StandbyModeTargetVoltage	129
6.5.9	DCDC_SetVDD1P8StandbyModeTargetVoltage	129
6.5.10	DCDC_GetVDD1P8StandbyModeTargetVoltage	129
6.5.11	DCDC_SetVDD1P0BuckModeTargetVoltage	130
6.5.12	DCDC_GetVDD1P0BuckModeTargetVoltage	131
6.5.13	DCDC_SetVDD1P8BuckModeTargetVoltage	131
6.5.14	DCDC_GetVDD1P8BuckModeTargetVoltage	131
6.5.15	DCDC_EnableVDD1P0TargetVoltageStepping	131
6.5.16	DCDC_EnableVDD1P8TargetVoltageStepping	132
6.5.17	DCDC_GetDefaultDetectionConfig	132
6.5.18	DCDC_SetDetectionConfig	132
6.5.19	DCDC_EnableOutputRangeComparator	133
6.5.20	DCDC_SetClockSource	133
6.5.21	DCDC_GetDefaultLowPowerConfig	133
6.5.22	DCDC_SetLowPowerConfig	134
6.5.23	DCDC_SetBandgapVoltageTrimValue	134
6.5.24	DCDC_GetDefaultLoopControlConfig	134
6.5.25	DCDC_SetLoopControlConfig	135
6.5.26	DCDC_SetMinPowerConfig	136
6.5.27	DCDC_SetLPComparatorBiasValue	136
6.5.28	DCDC_SetInternalRegulatorConfig	136
6.5.29	DCDC_EnableAdjustDelay	136
6.5.30	DCDC_EnableImproveTransition	137
6.5.31	DCDC_SetPointInit	137
6.5.32	DCDC_SetPointDeinit	137
6.5.33	DCDC_GetStatusFlags	138
6.5.34	DCDC_BootIntoDCM	138
6.5.35	DCDC_BootIntoCCM	138

Chapter 7 ROMAPI Driver

7.1	Overview	140
7.2	Data Structure Documentation	145
7.2.1	struct serial_nor_config_option_t	145
7.2.2	struct flexspi_lut_seq_t	145
7.2.3	struct flexspi_dll_time_t	145

Section No.	Title	Page No.
7.2.4	struct flexspi_mem_config_t	145
7.2.5	struct flexspi_nor_config_t	147
7.2.6	struct flexspi_xfer_t	148
7.3	Macro Definition Documentation	148
7.3.1	FSL_ROM_ROMAPI_VERSION	148
7.3.2	FSL_ROM_FLEXSPINOR_DRIVER_VERSION	148
7.3.3	FSL_ROM_HAS_FLEXSPINOR_API	148
7.3.4	FSL_ROM_HAS_RUNBOOTLOADER_API	148
7.3.5	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG	148
7.3.6	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT	148
7.3.7	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE	149
7.3.8	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_SECTOR	149
7.3.9	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_BLOCK	149
7.3.10	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_ALL	149
7.3.11	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_READ	149
7.3.12	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_UPDATE_LUT	149
7.3.13	FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_CMD_XFER	149
7.3.14	kROM_StatusGroup_FLEXSPINOR	149
7.3.15	FLEXSPI_CFG_BLK_TAG	149
7.3.16	NOR_CMD_LUT_SEQ_IDX_READ	149
7.4	Enumeration Type Documentation	149
7.4.1	anonymous enum	149
7.4.2	anonymous enum	150
7.4.3	_flexspi_read_sample_clk	150
7.4.4	anonymous enum	150
7.4.5	anonymous enum	150
7.4.6	_flexspi_serial_clk_freq	150
7.4.7	anonymous enum	151
7.4.8	anonymous enum	151
7.4.9	_flexspi_nor_status	151
7.4.10	flexspi_operation_t	152
7.5	Function Documentation	152
7.5.1	ROM_API_Init	152
7.5.2	ROM_RunBootloader	152
7.5.3	ROM_FLEXSPI_NorFlash_GetConfig	152
7.5.4	ROM_FLEXSPI_NorFlash_Init	153
7.5.5	ROM_FLEXSPI_NorFlash_ProgramPage	153
7.5.6	ROM_FLEXSPI_NorFlash_Read	154
7.5.7	ROM_FLEXSPI_NorFlash_Erase	155
7.5.8	ROM_FLEXSPI_NorFlash_EraseSector	156
7.5.9	ROM_FLEXSPI_NorFlash_EraseBlock	157
7.5.10	ROM_FLEXSPI_NorFlash_EraseAll	158

Section No.	Title	Page No.
7.5.11	ROM_FLEXSPI_NorFlash_CommandXfer	158
7.5.12	ROM_FLEXSPI_NorFlash_UpdateLut	159
7.5.13	ROM_FLEXSPI_NorFlash_WaitBusy	159
7.5.14	ROM_FLEXSPI_NorFlash_ClearCache	160

Chapter 8 IOMUXC: IOMUX Controller

8.1	Overview	161
8.2	Macro Definition Documentation	211
8.2.1	FSL_IOMUXC_DRIVER_VERSION	211
8.3	Function Documentation	211
8.3.1	IOMUXC_SetPinMux	212
8.3.2	IOMUXC_SetPinConfig	212
8.3.3	IOMUXC_SetSaiMClkClockSource	213
8.3.4	IOMUXC_MQSEnterSoftwareReset	213
8.3.5	IOMUXC_MQSEnable	213
8.3.6	IOMUXC_MQSConfig	214

Chapter 9 NIC301 Driver

Chapter 10 PGMC Driver

Chapter 11 PMU Driver

11.1	Overview	217
11.2	Data Structure Documentation	223
11.2.1	struct pmu_static_lpsr_ana_ldo_config_t	224
11.2.2	struct pmu_static_lpsr_dig_config_t	224
11.2.3	struct pmu_snvs_dig_config_t	225
11.2.4	struct pmu_static_bandgap_config_t	226
11.2.5	union pmu_well_bias_option_t	226
11.2.6	struct pmu_well_bias_config_t	227
11.2.7	struct pmu_gpc_body_bias_config_t	228
11.3	Macro Definition Documentation	228
11.3.1	FSL_PMU_DRIVER_VERSION	228
11.4	Enumeration Type Documentation	228
11.4.1	_pmu_setpoint_map	228
11.4.2	pmu_ldo_name_t	229
11.4.3	pmu_body_bias_name_t	229
11.4.4	pmu_control_mode_t	229

Section No.	Title	Page No.
11.4.5	pmu_ldo_operate_mode_t	229
11.4.6	pmu_lpsr_ana_ldo_charge_pump_current_t	230
11.4.7	pmu_lpsr_ana_ldo_output_range_t	230
11.4.8	pmu_lpsr_dig_voltage_step_time_t	230
11.4.9	pmu_lpsr_dig_target_output_voltage_t	230
11.4.10	pmu_snvs_dig_charge_pump_current_t	231
11.4.11	pmu_snvs_dig_discharge_resistor_value_t	231
11.4.12	_pmu_static_bandgap_power_down_option	232
11.4.13	pmu_bandgap_output_VBG_voltage_value_t	232
11.4.14	pmu_bandgap_output_current_value_t	232
11.4.15	pmu_well_bias_power_source_t	232
11.4.16	pmu_bias_area_size_t	233
11.4.17	pmu_well_bias_typical_freq_t	233
11.4.18	pmu_adaptive_clock_source_t	233
11.4.19	pmu_freq_reduction_t	233
11.4.20	pmu_well_bias_1P8_adjustment_t	234
11.5	Function Documentation	234
11.5.1	PMU_SetPllLdoControlMode	234
11.5.2	PMU_SwitchPllLdoToGPCMode	234
11.5.3	PMU_StaticEnablePllLdo	234
11.5.4	PMU_SetLpsrAnaLdoControlMode	235
11.5.5	PMU_StaticEnableLpsrAnaLdoBypassMode	235
11.5.6	PMU_StaticCheckLpsrAnaLdoBypassMode	235
11.5.7	PMU_StaticGetLpsrAnaLdoDefaultConfig	236
11.5.8	PMU_StaticLpsrAnaLdoInit	236
11.5.9	PMU_StaticLpsrAnaLdoDeinit	236
11.5.10	PMU_SetLpsrDigLdoControlMode	236
11.5.11	PMU_StaticEnableLpsrDigLdoBypassMode	237
11.5.12	PMU_StaticCheckLpsrDigLdoBypassMode	237
11.5.13	PMU_StaticGetLpsrDigLdoDefaultConfig	237
11.5.14	PMU_StaticLpsrDigLdoInit	238
11.5.15	PMU_StaticLpsrDigLdoDeinit	238
11.5.16	PMU_GPCSetLpsrDigLdoTargetVoltage	238
11.5.17	PMU_GetSnvsDigLdoDefaultConfig	238
11.5.18	PMU_SnvsDigLdoInit	239
11.5.19	PMU_GPCEnableLdo	239
11.5.20	PMU_GPCSetLdoOperateMode	239
11.5.21	PMU_GPCEnableLdoTrackingMode	240
11.5.22	PMU_GPCEnableLdoBypassMode	240
11.5.23	PMU_GPCEnableLdoStandbyMode	240
11.5.24	PMU_SetBandgapControlMode	240
11.5.25	PMU_SwitchBandgapToGPCMode	241
11.5.26	PMU_DisableBandgapSelfBiasAfterPowerUp	241
11.5.27	PMU_EnableBandgapSelfBiasBeforePowerDown	241

Section No.	Title	Page No.
11.5.28	PMU_StaticBandgapInit	241
11.5.29	PMU_GPCEnableBandgap	241
11.5.30	PMU_GPCEnableBandgapStandbyMode	242
11.5.31	PMU_WellBiasInit	242
11.5.32	PMU_GetWellBiasDefaultConfig	242
11.5.33	PMU_SetBodyBiasControlMode	242
11.5.34	PMU_EnableBodyBias	243
11.5.35	PMU_GPCEnableBodyBias	243
11.5.36	PMU_GPCEnableBodyBiasStandbyMode	243
11.5.37	PMU_GPCGetBodyBiasDefaultConfig	244
11.5.38	PMU_GPCSetBodyBiasConfig	244
 Chapter 12 SOC_SRC Driver		
12.1	Overview	245
12.2	Data Structure Documentation	250
12.2.1	struct src_setpoint_authentication_t	250
12.2.2	struct src_domain_mode_authentication_t	251
12.3	Macro Definition Documentation	251
12.3.1	FSL_SRC_DRIVER_VERSION	251
12.4	Enumeration Type Documentation	251
12.4.1	src_core_name_t	252
12.4.2	src_boot_fuse_selection_t	252
12.4.3	src_global_system_reset_source_t	252
12.4.4	_src_global_system_reset_status_flags	252
12.4.5	src_global_system_reset_mode_t	253
12.4.6	src_reset_slice_name_t	253
12.4.7	_src_domain_mode_selection	254
12.4.8	_src_setpoint_selection	254
12.4.9	src_general_purpose_register_index_t	255
12.4.10	_src_slice_reset_source	255
12.4.11	src_slice_reset_state_t	255
12.5	Function Documentation	255
12.5.1	SRC_ReleaseCoreReset	256
12.5.2	SRC_GetBootConfig	257
12.5.3	SRC_GetBootMode	257
12.5.4	SRC_GetBootFuseSelection	257
12.5.5	SRC_GetSECCongfigFuseState	258
12.5.6	SRC_SetGlobalSystemResetMode	259
12.5.7	SRC_GetResetStatusFlags	259
12.5.8	SRC_ClearGlobalSystemResetStatus	259

Section No.	Title	Page No.
12.5.9	SRC_AssertSliceSoftwareReset	260
12.5.10	SRC_AllowUserModeAccess	260
12.5.11	SRC_AllowNonSecureModeAccess	260
12.5.12	SRC_LockAccessSetting	261
12.5.13	SRC_SetDomainIdWhiteList	261
12.5.14	SRC_LockDomainIdWhiteList	261
12.5.15	SRC_SetAssignList	262
12.5.16	SRC_LockAssignList	262
12.5.17	SRC_EnableSetPointTransferReset	262
12.5.18	SRC_EnableDomainModeTransferReset	263
12.5.19	SRC_SetSliceSetPointConfig	263
12.5.20	SRC_SetSliceDomainModeConfig	263
12.5.21	SRC_LockSliceMode	263
12.5.22	SRC_GetSliceResetStatusFlags	264
12.5.23	SRC_ClearSliceResetStatusFlags	264
12.5.24	SRC_GetSliceResetState	264
12.5.25	SRC_SetGeneralPurposeRegister	265
12.5.26	SRC_GetGeneralPurposeRegister	265

Chapter 13 ACMP: Analog Comparator Driver

13.1	Overview	266
13.2	Typical use case	266
13.2.1	Normal Configuration	266
13.2.2	Interrupt Configuration	266
13.2.3	Round robin Configuration	266
13.3	Data Structure Documentation	269
13.3.1	struct acmp_config_t	269
13.3.2	struct acmp_channel_config_t	270
13.3.3	struct acmp_filter_config_t	270
13.3.4	struct acmp_dac_config_t	271
13.3.5	struct acmp_round_robin_config_t	271
13.3.6	struct acmp_discrete_mode_config_t	272
13.4	Macro Definition Documentation	272
13.4.1	FSL_ACMP_DRIVER_VERSION	272
13.4.2	CMP_C0_CFx_MASK	272
13.5	Enumeration Type Documentation	272
13.5.1	_acmp_interrupt_enable	273
13.5.2	_acmp_status_flags	273
13.5.3	acmp_hysteresis_mode_t	273
13.5.4	acmp_reference_voltage_source_t	273

Section No.	Title	Page No.
13.5.5	acmp_fixed_port_t	273
13.5.6	acmp_dac_work_mode_t	274
13.5.7	acmp_discrete_clock_source_t	274
13.5.8	acmp_discrete_sample_time_t	274
13.5.9	acmp_discrete_phase_time_t	274
13.6	Function Documentation	275
13.6.1	ACMP_Init	275
13.6.2	ACMP_Deinit	275
13.6.3	ACMP_GetDefaultConfig	275
13.6.4	ACMP_Enable	275
13.6.5	ACMP_EnableLinkToDAC	276
13.6.6	ACMP_SetChannelConfig	276
13.6.7	ACMP_EnableDMA	276
13.6.8	ACMP_EnableWindowMode	277
13.6.9	ACMP_SetFilterConfig	277
13.6.10	ACMP_SetDACConfig	277
13.6.11	ACMP_SetRoundRobinConfig	278
13.6.12	ACMP_SetRoundRobinPreState	278
13.6.13	ACMP_GetRoundRobinStatusFlags	278
13.6.14	ACMP_ClearRoundRobinStatusFlags	279
13.6.15	ACMP_GetRoundRobinResult	279
13.6.16	ACMP_EnableInterrupts	279
13.6.17	ACMP_DisableInterrupts	280
13.6.18	ACMP_GetStatusFlags	280
13.6.19	ACMP_ClearStatusFlags	280
13.6.20	ACMP_SetDiscreteModeConfig	280
13.6.21	ACMP_GetDefaultDiscreteModeConfig	281
Chapter 14	ADC_ETC: ADC External Trigger Control	
14.1	Overview	282
14.2	Typical use case	282
14.2.1	Software trigger Configuration	282
14.2.2	Hardware trigger Configuration	282
14.3	Data Structure Documentation	283
14.3.1	struct adc_etc_config_t	283
14.3.2	struct adc_etc_trigger_chain_config_t	283
14.3.3	struct adc_etc_trigger_config_t	283
14.4	Macro Definition Documentation	283
14.4.1	FSL_ADC_ETC_DRIVER_VERSION	284
14.4.2	ADC_ETC_DMA_CTRL_TRGn_REQ_MASK	284

Section No.	Title	Page No.
14.5	Function Documentation	284
14.5.1	ADC_ETC_Init	284
14.5.2	ADC_ETC_Deinit	284
14.5.3	ADC_ETC_GetDefaultConfig	284
14.5.4	ADC_ETC_SetTriggerConfig	285
14.5.5	ADC_ETC_SetTriggerChainConfig	285
14.5.6	ADC_ETC_GetInterruptStatusFlags	285
14.5.7	ADC_ETC_ClearInterruptStatusFlags	286
14.5.8	ADC_ETC_EnableDMA	286
14.5.9	ADC_ETC_DisableDMA	286
14.5.10	ADC_ETC_GetDMAStatusFlags	286
14.5.11	ADC_ETC_ClearDMAStatusFlags	287
14.5.12	ADC_ETC_DoSoftwareReset	287
14.5.13	ADC_ETC_DoSoftwareTrigger	287
14.5.14	ADC_ETC_GetADCCConversionValue	288
 Chapter 15 AIPSTZ: AHB to IP Bridge		
15.1	Overview	289
15.2	Enumeration Type Documentation	289
15.2.1	aipstz_master_privilege_level_t	289
15.2.2	aipstz_master_t	290
15.2.3	aipstz_peripheral_access_control_t	290
15.2.4	aipstz_peripheral_t	290
15.3	Function Documentation	290
15.3.1	AIPSTZ_SetMasterPriviledgeLevel	290
15.3.2	AIPSTZ_SetPeripheralAccessControl	290
 Chapter 16 AOI: Crossbar AND/OR/INVERT Driver		
16.1	Overview	292
16.2	Function groups	292
16.2.1	AOI Initialization	292
16.2.2	AOI Get Set Operation	292
16.3	Typical use case	292
16.4	Data Structure Documentation	294
16.4.1	struct aoi_event_config_t	294
16.5	Macro Definition Documentation	295
16.5.1	FSL_AOI_DRIVER_VERSION	295

Section No.	Title	Page No.
16.6	Enumeration Type Documentation	295
16.6.1	aoi_input_config_t	295
16.6.2	aoi_event_t	295
16.7	Function Documentation	295
16.7.1	AOI_Init	295
16.7.2	AOI_Deinit	295
16.7.3	AOI_GetEventLogicConfig	296
16.7.4	AOI_SetEventLogicConfig	296
Chapter 17 ASRC: Asynchronous sample rate converter		
17.1	Overview	298
17.2	ASRC Driver	299
17.2.1	Overview	299
17.2.2	Data Structure Documentation	304
17.2.3	Macro Definition Documentation	307
17.2.4	Enumeration Type Documentation	307
17.2.5	Function Documentation	310
Chapter 18 CAAM: Cryptographic Acceleration and Assurance Module		
18.1	Overview	320
18.2	CAAM Driver Initialization and Configuration	320
18.3	Comments about API usage in RTOS	321
18.4	Comments about API usage in interrupt handler	321
18.5	Comments about DCACHE	321
18.6	CAAM Driver Examples	321
18.6.1	Simple examples	321
18.7	Data Structure Documentation	323
18.7.1	struct caam_job_callback_t	323
18.7.2	struct caam_handle_t	323
18.7.3	struct caam_config_t	324
18.8	Macro Definition Documentation	324
18.8.1	FSL_CAAM_DRIVER_VERSION	324
18.9	Enumeration Type Documentation	325
18.9.1	anonymous enum	325

Section No.	Title	Page No.
18.9.2	caam_job_ring_t	325
18.9.3	caam_wait_mode_t	326
18.9.4	caam_rng_sample_mode_t	326
18.9.5	caam_rng_ring_osc_div_t	326
18.9.6	caam_priblob_t	326
18.9.7	caam_ext_key_xfr_source_t	327
18.10	Function Documentation	327
18.10.1	CAAM_Init	327
18.10.2	CAAM_Deinit	327
18.10.3	CAAM_GetDefaultConfig	327
18.10.4	CAAM_Wait	328
18.10.5	CAAM_ExternalKeyTransfer	328
18.11	CAAM Blocking APIs	330
18.11.1	Overview	330
18.11.2	CAAM RNG driver	331
18.11.3	CAAM DES driver	336
18.11.4	CAAM AES driver	356
18.11.5	CAAM HASH driver	365
18.11.6	CAAM PKHA driver	369
18.12	CAAM Non-blocking APIs	385
18.12.1	Overview	385
18.12.2	CAAM Non-blocking DES driver	386
18.12.3	CAAM Non-blocking HASH driver	406
18.12.4	CAAM Non-blocking RNG driver	409
18.12.5	CAAM Non-blocking AES driver	410
Chapter 19 CACHE: ARMV7-M7 CACHE Memory Controller		
19.1	Overview	419
19.2	Function groups	419
19.2.1	L1 CACHE Operation	419
19.2.2	L2 CACHE Operation	419
19.3	Macro Definition Documentation	420
19.3.1	FSL_CACHE_DRIVER_VERSION	420
19.4	Function Documentation	420
19.4.1	L1CACHE_InvalidateICacheByRange	421
19.4.2	L1CACHE_InvalidateDCacheByRange	422
19.4.3	L1CACHE_CleanDCacheByRange	422
19.4.4	L1CACHE_CleanInvalidateDCacheByRange	423
19.4.5	ICACHE_InvalidateByRange	424

Section No.	Title	Page No.
19.4.6	DCACHE_InvalidateByRange	424
19.4.7	DCACHE_CleanByRange	425
19.4.8	DCACHE_CleanInvalidateByRange	426

Chapter 20 CACHE: LMEM CACHE Memory Controller

20.1	Overview	427
20.2	Function groups	427
20.2.1	L1 CACHE Operation	427
20.3	Macro Definition Documentation	429
20.3.1	FSL_CACHE_DRIVER_VERSION	429
20.3.2	L1CODEBUSCACHE_LINESIZE_BYTE	429
20.3.3	L1SYSTEMBUSCACHE_LINESIZE_BYTE	429
20.4	Function Documentation	429
20.4.1	L1CACHE_InvalidateCodeCacheByRange	429
20.4.2	L1CACHE_CleanCodeCacheByRange	429
20.4.3	L1CACHE_CleanInvalidateCodeCacheByRange	430
20.4.4	L1CACHE_EnableCodeCacheWriteBuffer	430
20.4.5	L1CACHE_InvalidateSystemCacheByRange	430
20.4.6	L1CACHE_CleanSystemCacheByRange	431
20.4.7	L1CACHE_CleanInvalidateSystemCacheByRange	431
20.4.8	L1CACHE_EnableSystemCacheWriteBuffer	432
20.4.9	L1CACHE_InvalidateICacheByRange	433
20.4.10	L1CACHE_InvalidateDCacheByRange	433
20.4.11	L1CACHE_CleanDCacheByRange	433
20.4.12	L1CACHE_CleanInvalidateDCacheByRange	434
20.4.13	ICACHE_InvalidateByRange	434
20.4.14	DCACHE_InvalidateByRange	434
20.4.15	DCACHE_CleanByRange	435
20.4.16	DCACHE_CleanInvalidateByRange	435

Chapter 21 Common Driver

21.1	Overview	436
21.2	Macro Definition Documentation	438
21.2.1	FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ	438
21.2.2	MAKE_STATUS	438
21.2.3	MAKE_VERSION	439
21.2.4	FSL_COMMON_DRIVER_VERSION	439
21.2.5	DEBUG_CONSOLE_DEVICE_TYPE_NONE	439
21.2.6	DEBUG_CONSOLE_DEVICE_TYPE_UART	439

Section No.	Title	Page No.
21.2.7	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	439
21.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	439
21.2.9	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	439
21.2.10	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	439
21.2.11	DEBUG_CONSOLE_DEVICE_TYPE_IUART	439
21.2.12	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	439
21.2.13	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	439
21.2.14	DEBUG_CONSOLE_DEVICE_TYPE_SWO	439
21.2.15	DEBUG_CONSOLE_DEVICE_TYPE_QSCI	439
21.2.16	ARRAY_SIZE	439
21.3	Typedef Documentation	439
21.3.1	status_t	439
21.4	Enumeration Type Documentation	440
21.4.1	_status_groups	440
21.4.2	anonymous enum	442
21.5	Function Documentation	442
21.5.1	SDK_Malloc	443
21.5.2	SDK_Free	444
21.5.3	SDK_DelayAtLeastUs	444
 Chapter 22 CSI: CMOS Sensor Interface		
22.1	Overview	445
22.2	Frame Buffer Queue	445
22.3	Fragment Mode	445
22.4	Typical use case	446
22.5	Data Structure Documentation	449
22.5.1	struct csi_config_t	449
22.5.2	struct _csi_handle	450
22.6	Macro Definition Documentation	451
22.6.1	CSI_DRIVER_QUEUE_SIZE	451
22.6.2	CSI_DRIVER_FRAG_MODE	451
22.7	Typedef Documentation	451
22.7.1	csi_transfer_callback_t	451
22.8	Enumeration Type Documentation	451
22.8.1	anonymous enum	451

Section No.	Title	Page No.
22.8.2	csi_work_mode_t	452
22.8.3	csi_data_bus_t	452
22.8.4	_csi_polarity_flags	452
22.8.5	csi_fifo_t	452
22.8.6	_csi_interrupt_enable	452
22.8.7	_csi_flags	453
22.9	Function Documentation	454
22.9.1	CSI_Init	454
22.9.2	CSI_Deinit	454
22.9.3	CSI_Reset	454
22.9.4	CSI_GetDefaultConfig	455
22.9.5	CSI_ClearFifo	455
22.9.6	CSI_ReflashFifoDma	455
22.9.7	CSI_EnableFifoDmaRequest	456
22.9.8	CSI_Start	456
22.9.9	CSI_Stop	456
22.9.10	CSI_SetRxBufferAddr	456
22.9.11	CSI_EnableInterrupts	457
22.9.12	CSI_DisableInterrupts	457
22.9.13	CSI_GetStatusFlags	457
22.9.14	CSI_ClearStatusFlags	457
22.9.15	CSI_TransferCreateHandle	458
22.9.16	CSI_TransferStart	458
22.9.17	CSI_TransferStop	459
22.9.18	CSI_TransferSubmitEmptyBuffer	459
22.9.19	CSI_TransferGetFullBuffer	460
22.9.20	CSI_TransferHandleIRQ	460
Chapter 23	DAC12: 12-bit Digital-to-Analog Converter Driver	
23.1	Overview	461
23.2	Typical use case	461
23.2.1	A simple use case to output the user-defined DAC12 value.	461
23.2.2	Working with the trigger	461
23.3	Data Structure Documentation	464
23.3.1	struct dac12_hardware_info_t	464
23.3.2	struct dac12_config_t	464
23.4	Macro Definition Documentation	465
23.4.1	FSL_DAC12_DRIVER_VERSION	465
23.4.2	DAC12_CR_W1C_FLAGS_MASK	465
23.4.3	DAC12_CR_ALL_FLAGS_MASK	465

Section No.	Title	Page No.
23.5	Enumeration Type Documentation	465
23.5.1	_dac12_status_flags	465
23.5.2	_dac12_interrupt_enable	466
23.5.3	dac12_fifo_size_info_t	466
23.5.4	dac12_fifo_work_mode_t	466
23.5.5	dac12_reference_voltage_source_t	466
23.5.6	dac12_fifo_trigger_mode_t	467
23.5.7	dac12_reference_current_source_t	467
23.5.8	dac12_speed_mode_t	467
23.6	Function Documentation	467
23.6.1	DAC12_GetHardwareInfo	467
23.6.2	DAC12_Init	468
23.6.3	DAC12_GetDefaultConfig	468
23.6.4	DAC12_Deinit	468
23.6.5	DAC12_Enable	468
23.6.6	DAC12_ResetConfig	469
23.6.7	DAC12_ResetFIFO	469
23.6.8	DAC12_GetStatusFlags	469
23.6.9	DAC12_ClearStatusFlags	469
23.6.10	DAC12_EnableInterrupts	470
23.6.11	DAC12_DisableInterrupts	470
23.6.12	DAC12_EnableDMA	470
23.6.13	DAC12_SetData	470
23.6.14	DAC12_DoSoftwareTrigger	471
23.6.15	DAC12_GetFIFOReadPointer	471
23.6.16	DAC12_GetFIFOWritePointer	471
Chapter 24	DMAMUX: Direct Memory Access Multiplexer Driver	
24.1	Overview	472
24.2	Typical use case	472
24.2.1	DMAMUX Operation	472
24.3	Macro Definition Documentation	472
24.3.1	FSL_DMAMUX_DRIVER_VERSION	472
24.4	Function Documentation	473
24.4.1	DMAMUX_Init	473
24.4.2	DMAMUX_Deinit	474
24.4.3	DMAMUX_EnableChannel	474
24.4.4	DMAMUX_DisableChannel	474
24.4.5	DMAMUX_SetSource	475
24.4.6	DMAMUX_EnablePeriodTrigger	475

Section No.	Title	Page No.
24.4.7	DMAMUX_DisablePeriodTrigger	475
24.4.8	DMAMUX_EnableAlwaysOn	475
Chapter 25 eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver		
25.1	Overview	477
25.2	Typical use case	477
25.2.1	eDMA Operation	477
25.3	Data Structure Documentation	482
25.3.1	struct edma_config_t	482
25.3.2	struct edma_transfer_config_t	483
25.3.3	struct edma_channel_Preemption_config_t	484
25.3.4	struct edma_minor_offset_config_t	484
25.3.5	struct edma_tcd_t	484
25.3.6	struct edma_handle_t	485
25.4	Macro Definition Documentation	486
25.4.1	FSL_EDMA_DRIVER_VERSION	486
25.5	Typedef Documentation	486
25.5.1	edma_callback	486
25.6	Enumeration Type Documentation	487
25.6.1	edma_transfer_size_t	487
25.6.2	edma_modulo_t	487
25.6.3	edma_bandwidth_t	488
25.6.4	edma_channel_link_type_t	488
25.6.5	anonymous enum	488
25.6.6	anonymous enum	489
25.6.7	edma_interrupt_enable_t	489
25.6.8	edma_transfer_type_t	489
25.6.9	anonymous enum	489
25.7	Function Documentation	490
25.7.1	EDMA_Init	490
25.7.2	EDMA_Deinit	491
25.7.3	EDMA_InstallTCD	491
25.7.4	EDMA_GetDefaultConfig	491
25.7.5	EDMA_EnableContinuousChannelLinkMode	492
25.7.6	EDMA_EnableMinorLoopMapping	492
25.7.7	EDMA_ResetChannel	492
25.7.8	EDMA_SetTransferConfig	493
25.7.9	EDMA_SetMinorOffsetConfig	493
25.7.10	EDMA_SetChannelPreemptionConfig	494

Section No.	Title	Page No.
25.7.11	EDMA_SetChannelLink	494
25.7.12	EDMA_SetBandWidth	495
25.7.13	EDMA_SetModulo	495
25.7.14	EDMA_EnableAsyncRequest	496
25.7.15	EDMA_EnableAutoStopRequest	496
25.7.16	EDMA_EnableChannelInterrupts	496
25.7.17	EDMA_DisableChannelInterrupts	496
25.7.18	EDMA_SetMajorOffsetConfig	497
25.7.19	EDMA_TcdReset	497
25.7.20	EDMA_TcdSetTransferConfig	497
25.7.21	EDMA_TcdSetMinorOffsetConfig	498
25.7.22	EDMA_TcdSetChannelLink	498
25.7.23	EDMA_TcdSetBandWidth	499
25.7.24	EDMA_TcdSetModulo	499
25.7.25	EDMA_TcdEnableAutoStopRequest	500
25.7.26	EDMA_TcdEnableInterrupts	501
25.7.27	EDMA_TcdDisableInterrupts	501
25.7.28	EDMA_TcdSetMajorOffsetConfig	501
25.7.29	EDMA_EnableChannelRequest	501
25.7.30	EDMA_DisableChannelRequest	502
25.7.31	EDMA_TriggerChannelStart	502
25.7.32	EDMA_GetRemainingMajorLoopCount	502
25.7.33	EDMA_GetErrorStatusFlags	503
25.7.34	EDMA_GetChannelStatusFlags	503
25.7.35	EDMA_ClearChannelStatusFlags	503
25.7.36	EDMA_CreateHandle	504
25.7.37	EDMA_InstallTCDDMemory	504
25.7.38	EDMA_SetCallback	504
25.7.39	EDMA_PrepareTransferConfig	505
25.7.40	EDMA_PrepareTransfer	505
25.7.41	EDMA_SubmitTransfer	506
25.7.42	EDMA_StartTransfer	507
25.7.43	EDMA_StopTransfer	508
25.7.44	EDMA_AbortTransfer	508
25.7.45	EDMA_GetUnusedTCDNumber	508
25.7.46	EDMA_GetNextTCDAddress	508
25.7.47	EDMA_HandleIRQ	509

Chapter 26 eLCDIF: Enhanced LCD Interface

26.1	Overview	510
26.2	Typical use case	510
26.2.1	Frame buffer update	510
26.2.2	Alpha surface	510

Section No.	Title	Page No.
26.3	Data Structure Documentation	513
26.3.1	struct elcdif_pixel_format_reg_t	514
26.3.2	struct elcdif_rgb_mode_config_t	514
26.3.3	struct elcdif_as_buffer_config_t	515
26.3.4	struct elcdif_as_blend_config_t	516
26.4	Enumeration Type Documentation	516
26.4.1	_elcdif_polarity_flags	516
26.4.2	_elcdif_interrupt_enable	516
26.4.3	_elcdif_interrupt_flags	517
26.4.4	_elcdif_status_flags	517
26.4.5	elcdif_pixel_format_t	517
26.4.6	elcdif_lcd_data_bus_t	518
26.4.7	elcdif_as_pixel_format_t	518
26.4.8	elcdif_alpha_mode_t	518
26.4.9	elcdif_rop_mode_t	518
26.4.10	elcdif_lut_t	519
26.5	Function Documentation	519
26.5.1	ELCDIF_RgbModeInit	519
26.5.2	ELCDIF_RgbModeGetDefaultConfig	520
26.5.3	ELCDIF_Deinit	520
26.5.4	ELCDIF_RgbModeSetPixelFormat	520
26.5.5	ELCDIF_RgbModeStart	521
26.5.6	ELCDIF_RgbModeStop	521
26.5.7	ELCDIF_SetNextBufferAddr	521
26.5.8	ELCDIF_Reset	521
26.5.9	ELCDIF_GetCrcValue	521
26.5.10	ELCDIF_GetBusMasterErrorAddr	522
26.5.11	ELCDIF_GetStatus	522
26.5.12	ELCDIF_GetLFifoCount	523
26.5.13	ELCDIF_EnableInterrupts	523
26.5.14	ELCDIF_DisableInterrupts	523
26.5.15	ELCDIF_GetInterruptStatus	524
26.5.16	ELCDIF_ClearInterruptStatus	524
26.5.17	ELCDIF_EnableLut	524
26.5.18	ELCDIF_UpdateLut	524
Chapter 27	ENC: Quadrature Encoder/Decoder	
27.1	Overview	526
27.2	Function groups	526
27.2.1	Initialization and De-initialization	526
27.2.2	Status	526

Section No.	Title	Page No.
27.2.3	Interrupts	526
27.2.4	Value Operation	526
27.3	Typical use case	526
27.3.1	Polling Configuration	526
27.4	Data Structure Documentation	529
27.4.1	struct enc_config_t	529
27.4.2	struct enc_self_test_config_t	531
27.5	Enumeration Type Documentation	532
27.5.1	_enc_interrupt_enable	532
27.5.2	_enc_status_flags	532
27.5.3	_enc_signal_status_flags	532
27.5.4	enc_home_trigger_mode_t	533
27.5.5	enc_index_trigger_mode_t	533
27.5.6	enc_decoder_work_mode_t	533
27.5.7	enc_position_match_mode_t	534
27.5.8	enc_revolution_count_condition_t	534
27.5.9	enc_self_test_direction_t	534
27.5.10	enc_prescaler_t	534
27.6	Function Documentation	534
27.6.1	ENC_Init	534
27.6.2	ENC_Deinit	535
27.6.3	ENC_GetDefaultConfig	535
27.6.4	ENC_DoSoftwareLoadInitialPositionValue	536
27.6.5	ENC_SetSelfTestConfig	536
27.6.6	ENC_EnableWatchdog	536
27.6.7	ENC_SetInitialPositionValue	536
27.6.8	ENC_GetStatusFlags	537
27.6.9	ENC_ClearStatusFlags	537
27.6.10	ENC_GetSignalStatusFlags	537
27.6.11	ENC_EnableInterrupts	537
27.6.12	ENC_DisableInterrupts	538
27.6.13	ENC_GetEnabledInterrupts	538
27.6.14	ENC_GetPositionValue	538
27.6.15	ENC_GetHoldPositionValue	538
27.6.16	ENC_GetPositionDifferenceValue	539
27.6.17	ENC_GetHoldPositionDifferenceValue	539
27.6.18	ENC_GetRevolutionValue	539
27.6.19	ENC_GetHoldRevolutionValue	540
27.6.20	ENC_GetLastEdgeTimeValue	540
27.6.21	ENC_GetHoldLastEdgeTimeValue	540
27.6.22	ENC_GetPositionDifferencePeriodValue	541

Section No.	Title	Page No.
27.6.23	ENC_GetPositionDifferencePeriodBufferValue	541
27.6.24	ENC_GetHoldPositionDifferencePeriodValue	541
Chapter 28 ENET: Ethernet MAC Driver		
28.1	Overview	543
28.2	Operations of Ethernet MAC Driver	543
28.2.1	MII interface Operation	543
28.2.2	MAC address filter	543
28.2.3	Other Baisc control Operations	543
28.2.4	Transactional Operation	543
28.2.5	PTP IEEE 1588 Feature Operation	544
28.3	Typical use case	544
28.3.1	ENET Initialization, receive, and transmit operations	544
28.4	Data Structure Documentation	553
28.4.1	struct enet_rx_bd_struct_t	553
28.4.2	struct enet_tx_bd_struct_t	553
28.4.3	struct enet_data_error_stats_t	553
28.4.4	struct enet_rx_frame_error_t	554
28.4.5	struct enet_transfer_stats_t	554
28.4.6	struct enet_frame_info_t	555
28.4.7	struct enet_tx_dirty_ring_t	556
28.4.8	struct enet_buffer_config_t	556
28.4.9	struct enet_intcoalesce_config_t	558
28.4.10	struct enet_avb_config_t	558
28.4.11	struct enet_config_t	559
28.4.12	struct enet_tx_bd_ring_t	561
28.4.13	struct enet_rx_bd_ring_t	562
28.4.14	struct _enet_handle	562
28.5	Macro Definition Documentation	563
28.5.1	FSL_ENET_DRIVER_VERSION	563
28.5.2	ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK	563
28.5.3	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK	563
28.5.4	ENET_BUFFDESCRIPTOR_RX_WRAP_MASK	563
28.5.5	ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask	563
28.5.6	ENET_BUFFDESCRIPTOR_RX_LAST_MASK	563
28.5.7	ENET_BUFFDESCRIPTOR_RX_MISS_MASK	563
28.5.8	ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK	564
28.5.9	ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK	564
28.5.10	ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK	564
28.5.11	ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK	564

Section No.	Title	Page No.
28.5.12	ENET_BUFFDESCRIPTOR_RX_CRC_MASK	564
28.5.13	ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK	564
28.5.14	ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK	564
28.5.15	ENET_BUFFDESCRIPTOR_TX_READY_MASK	564
28.5.16	ENET_BUFFDESCRIPTOR_TX_SOFTWENER1_MASK	564
28.5.17	ENET_BUFFDESCRIPTOR_TX_WRAP_MASK	564
28.5.18	ENET_BUFFDESCRIPTOR_TX_SOFTWENER2_MASK	564
28.5.19	ENET_BUFFDESCRIPTOR_TX_LAST_MASK	564
28.5.20	ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK	564
28.5.21	ENET_BUFFDESCRIPTOR_RX_ERR_MASK	564
28.5.22	ENET_FRAME_MAX_FRAMELEN	564
28.5.23	ENET_FRAME_VLAN_TAGLEN	564
28.5.24	ENET_FRAME_CRC_LEN	565
28.5.25	ENET_FIFO_MIN_RX_FULL	565
28.5.26	ENET_RX_MIN_BUFFERSIZE	565
28.5.27	ENET_PHY_MAXADDRESS	565
28.5.28	ENET_TX_INTERRUPT	565
28.5.29	ENET_RX_INTERRUPT	565
28.5.30	ENET_TS_INTERRUPT	565
28.5.31	ENET_ERR_INTERRUPT	565
28.6	Typedef Documentation	566
28.6.1	enet_rx_alloc_callback_t	566
28.6.2	enet_rx_free_callback_t	566
28.6.3	enet_callback_t	566
28.6.4	enet_isr_ring_t	566
28.7	Enumeration Type Documentation	566
28.7.1	anonymous enum	566
28.7.2	enet_mii_mode_t	566
28.7.3	enet_mii_speed_t	566
28.7.4	enet_mii_duplex_t	567
28.7.5	enet_mii_write_t	567
28.7.6	enet_mii_read_t	567
28.7.7	enet_mii_extend_opcode	567
28.7.8	enet_special_control_flag_t	567
28.7.9	enet_interrupt_enable_t	568
28.7.10	enet_event_t	569
28.7.11	enet_idle_slope_t	569
28.7.12	enet_tx_accelerator_t	570
28.7.13	enet_rx_accelerator_t	570
28.8	Function Documentation	570
28.8.1	ENET_GetInstance	570
28.8.2	ENET_GetDefaultConfig	570

Section No.	Title	Page No.
28.8.3	ENET_Up	571
28.8.4	ENET_Init	572
28.8.5	ENET_Down	573
28.8.6	ENET_Deinit	573
28.8.7	ENET_Reset	573
28.8.8	ENET_SetMII	573
28.8.9	ENET_SetSMI	574
28.8.10	ENET_GetSMI	574
28.8.11	ENET_ReadSMIData	574
28.8.12	ENET_StartSMIRead	575
28.8.13	ENET_StartSMIWrite	575
28.8.14	ENET_StartExtC45SMIWriteReg	575
28.8.15	ENET_StartExtC45SMIWriteData	576
28.8.16	ENET_StartExtC45SMIReadData	576
28.8.17	ENET_SetRGMIIIClockDelay	577
28.8.18	ENET_SetMacAddr	577
28.8.19	ENET_GetMacAddr	577
28.8.20	ENET_AddMulticastGroup	578
28.8.21	ENET_LeaveMulticastGroup	578
28.8.22	ENET_ActiveRead	578
28.8.23	ENET_EnableSleepMode	579
28.8.24	ENET_GetAccelFunction	580
28.8.25	ENET_EnableInterrupts	580
28.8.26	ENET_DisableInterrupts	580
28.8.27	ENET_GetInterruptStatus	581
28.8.28	ENET_ClearInterruptStatus	581
28.8.29	ENET_SetRxISRHandler	581
28.8.30	ENET_SetTxISRHandler	582
28.8.31	ENET_SetErrISRHandler	582
28.8.32	ENET_SetCallback	582
28.8.33	ENET_GetRxErrBeforeReadFrame	583
28.8.34	ENET_GetStatistics	583
28.8.35	ENET_GetRxFrameSize	583
28.8.36	ENET_ReadFrame	584
28.8.37	ENET_SendFrame	585
28.8.38	ENET_SetTxReclaim	586
28.8.39	ENET_ReclaimTxDescriptor	586
28.8.40	ENET_GetRxBuffer	587
28.8.41	ENET_ReleaseRxBuffer	588
28.8.42	ENET_GetRxFrame	588
28.8.43	ENET_StartTxFrame	589
28.8.44	ENET_SendFrameZeroCopy	590
28.8.45	ENET_TransmitIRQHandler	591
28.8.46	ENET_ReceiveIRQHandler	591
28.8.47	ENET_CommonFrame1IRQHandler	591

Section No.	Title	Page No.
28.8.48	ENET_CommonFrame2IRQHandler	592
28.8.49	ENET_ErrorIRQHandler	593
28.8.50	ENET_Ptp1588IRQHandler	593
28.8.51	ENET_CommonFrame0IRQHandler	593
28.9	Variable Documentation	593
28.9.1	s_enetClock	593
28.10	ENET CMSIS Driver	594
28.10.1	Typical use case	594
 Chapter 29 EQOS-TSN: Ethernet QoS with TSN Driver		
29.1	EQOS-TSN: Ethernet QoS with TSN Driver	596
29.1.1	Initialize and De-initialize interface Operation	596
29.1.2	Other basic operation	596
29.1.3	Interrupt operation	596
29.1.4	Functional Operation	596
29.1.5	Transactional Operation	597
 Chapter 30 EWM: External Watchdog Monitor Driver		
30.1	Overview	598
30.2	Typical use case	598
30.3	Data Structure Documentation	599
30.3.1	struct ewm_config_t	599
30.4	Macro Definition Documentation	599
30.4.1	FSL_EWM_DRIVER_VERSION	599
30.5	Enumeration Type Documentation	599
30.5.1	ewm_lpo_clock_source_t	599
30.5.2	_ewm_interrupt_enable_t	600
30.5.3	_ewm_status_flags_t	600
30.6	Function Documentation	600
30.6.1	EWM_Init	600
30.6.2	EWM_Deinit	600
30.6.3	EWM_GetDefaultConfig	601
30.6.4	EWM_EnableInterrupts	601
30.6.5	EWM_DisableInterrupts	601
30.6.6	EWM_GetStatusFlags	602
30.6.7	EWM_Refresh	602

Section No.	Title	Page No.
Chapter 31 FlexCAN: Flex Controller Area Network Driver		
31.1	Overview	604
31.2	FlexCAN Driver	605
31.2.1	Overview	605
31.2.2	Typical use case	605
31.2.3	Data Structure Documentation	615
31.2.4	Macro Definition Documentation	622
31.2.5	Enumeration Type Documentation	627
31.2.6	Function Documentation	633
31.3	FlexCAN eDMA Driver	657
31.3.1	Overview	657
31.3.2	Data Structure Documentation	657
31.3.3	Macro Definition Documentation	658
31.3.4	Typedef Documentation	658
31.3.5	Function Documentation	658
Chapter 32 FlexIO: FlexIO Driver		
32.1	Overview	662
32.2	FlexIO Driver	663
32.2.1	Overview	663
32.2.2	Data Structure Documentation	667
32.2.3	Macro Definition Documentation	670
32.2.4	Typedef Documentation	670
32.2.5	Enumeration Type Documentation	670
32.2.6	Function Documentation	675
32.2.7	Variable Documentation	685
32.3	FlexIO Camera Driver	686
32.3.1	Overview	686
32.3.2	Typical use case	686
32.3.3	Data Structure Documentation	689
32.3.4	Macro Definition Documentation	690
32.3.5	Enumeration Type Documentation	690
32.3.6	Function Documentation	691
32.3.7	FlexIO eDMA Camera Driver	694
32.4	FlexIO I2C Master Driver	698
32.4.1	Overview	698
32.4.2	Typical use case	698
32.4.3	Data Structure Documentation	702
32.4.4	Macro Definition Documentation	704

Section No.	Title	Page No.
32.4.5	Typedef Documentation	704
32.4.6	Enumeration Type Documentation	705
32.4.7	Function Documentation	705
32.5	FlexIO I2S Driver	715
32.5.1	Overview	715
32.5.2	Typical use case	715
32.5.3	Data Structure Documentation	720
32.5.4	Macro Definition Documentation	722
32.5.5	Enumeration Type Documentation	723
32.5.6	Function Documentation	724
32.5.7	FlexIO eDMA I2S Driver	735
32.6	FlexIO MCU Interface LCD Driver	741
32.6.1	Overview	741
32.6.2	Typical use case	741
32.6.3	Data Structure Documentation	747
32.6.4	Macro Definition Documentation	750
32.6.5	Typedef Documentation	751
32.6.6	Enumeration Type Documentation	751
32.6.7	Function Documentation	752
32.6.8	FlexIO eDMA MCU Interface LCD Driver	765
32.7	FlexIO SPI Driver	771
32.7.1	Overview	771
32.7.2	Typical use case	771
32.7.3	Data Structure Documentation	778
32.7.4	Macro Definition Documentation	781
32.7.5	Typedef Documentation	781
32.7.6	Enumeration Type Documentation	782
32.7.7	Function Documentation	783
32.7.8	FlexIO eDMA SPI Driver	797
32.8	FlexIO UART Driver	803
32.8.1	Overview	803
32.8.2	Typical use case	803
32.8.3	Data Structure Documentation	812
32.8.4	Macro Definition Documentation	814
32.8.5	Typedef Documentation	814
32.8.6	Enumeration Type Documentation	815
32.8.7	Function Documentation	816
32.8.8	FlexIO eDMA UART Driver	827

Section No.	Title	Page No.
Chapter 33 FLEXRAM: on-chip RAM manager		
33.1	Overview	833
33.2	Data Structure Documentation	835
33.2.1	struct flexram_ocram_ecc_single_error_info_t	836
33.2.2	struct flexram_ocram_ecc_multi_error_info_t	836
33.2.3	struct flexram_itcm_ecc_single_error_info_t	836
33.2.4	struct flexram_itcm_ecc_multi_error_info_t	837
33.2.5	struct flexram_dtcmm_ecc_single_error_info_t	837
33.2.6	struct flexram_dtcmm_ecc_multi_error_info_t	837
33.3	Macro Definition Documentation	838
33.3.1	FSL_FLEXRAM_DRIVER_VERSION	838
33.3.2	FLEXRAM_ECC_ERROR_DETAILED_INFO	838
33.4	Enumeration Type Documentation	838
33.4.1	anonymous enum	838
33.4.2	anonymous enum	838
33.4.3	flexram_tcm_access_mode_t	838
33.4.4	anonymous enum	839
33.5	Function Documentation	839
33.5.1	FLEXRAM_Init	839
33.5.2	FLEXRAM_GetInterruptStatus	839
33.5.3	FLEXRAM_ClearInterruptStatus	839
33.5.4	FLEXRAM_EnableInterruptStatus	839
33.5.5	FLEXRAM_DisableInterruptStatus	840
33.5.6	FLEXRAM_EnableInterruptSignal	840
33.5.7	FLEXRAM_DisableInterruptSignal	840
33.5.8	FLEXRAM_SetTCMReadAccessMode	840
33.5.9	FLEXRAM_SetTCMWriteAccessMode	841
33.5.10	FLEXRAM_EnableForceRamClockOn	841
33.5.11	FLEXRAM_SetOCRAMMagicAddr	841
33.5.12	FLEXRAM_SetDTCCMMagicAddr	842
33.5.13	FLEXRAM_SetITCCMMagicAddr	842
33.5.14	FLEXRAM_EnableECC	842
33.5.15	FLEXRAM_GetOcramSingleErroInfo	843
33.5.16	FLEXRAM_GetOcramMultiErroInfo	843
33.5.17	FLEXRAM_GetItcmSingleErroInfo	843
33.5.18	FLEXRAM_GetItcmMultiErroInfo	843
33.5.19	FLEXRAM_GetDtcmmSingleErroInfo	844
33.5.20	FLEXRAM_GetDtcmmMultiErroInfo	844

Section No.	Title	Page No.
Chapter 34 FLEXSPI: Flexible Serial Peripheral Interface Driver		
34.1	Overview	845
34.2	Data Structure Documentation	850
34.2.1	struct flexspi_config_t	850
34.2.2	struct flexspi_device_config_t	853
34.2.3	struct flexspi_transfer_t	854
34.2.4	struct _flexspi_handle	855
34.3	Macro Definition Documentation	855
34.3.1	FSL_FLEXSPI_DRIVER_VERSION	855
34.3.2	FLEXSPI_LUT_SEQ	855
34.4	Typedef Documentation	856
34.4.1	flexspi_transfer_callback_t	856
34.5	Enumeration Type Documentation	856
34.5.1	anonymous enum	856
34.5.2	anonymous enum	856
34.5.3	flexspi_pad_t	857
34.5.4	flexspi_flags_t	857
34.5.5	flexspi_read_sample_clock_t	858
34.5.6	flexspi_cs_interval_cycle_unit_t	858
34.5.7	flexspi_ahb_write_wait_unit_t	858
34.5.8	flexspi_ip_error_code_t	859
34.5.9	flexspi_ahb_error_code_t	859
34.5.10	flexspi_port_t	859
34.5.11	flexspi_arb_command_source_t	859
34.5.12	flexspi_command_type_t	860
34.6	Function Documentation	860
34.6.1	FLEXSPI_GetInstance	860
34.6.2	FLEXSPI_CheckAndClearError	860
34.6.3	FLEXSPI_Init	860
34.6.4	FLEXSPI_GetDefaultConfig	860
34.6.5	FLEXSPI_Deinit	861
34.6.6	FLEXSPI_UpdateDIIValue	861
34.6.7	FLEXSPI_SetFlashConfig	861
34.6.8	FLEXSPI_SoftwareReset	862
34.6.9	FLEXSPI_Enable	863
34.6.10	FLEXSPI_EnableInterrupts	863
34.6.11	FLEXSPI_DisableInterrupts	863
34.6.12	FLEXSPI_EnableTxDMA	863
34.6.13	FLEXSPI_EnableRxDMA	864
34.6.14	FLEXSPI_GetTxFifoAddress	864

Section No.	Title	Page No.
34.6.15	FLEXSPI_GetRxFifoAddress	864
34.6.16	FLEXSPI_ResetFifos	865
34.6.17	FLEXSPI_GetFifoCounts	866
34.6.18	FLEXSPI_GetInterruptStatusFlags	866
34.6.19	FLEXSPI_ClearInterruptStatusFlags	866
34.6.20	FLEXSPI_GetArbitratorCommandSource	867
34.6.21	FLEXSPI_GetIPCommandErrorCode	868
34.6.22	FLEXSPI_GetAHBCommandErrorCode	868
34.6.23	FLEXSPI_GetBusIdleStatus	868
34.6.24	FLEXSPI_UpdateRxSampleClock	869
34.6.25	FLEXSPI_EnableIPParallelMode	869
34.6.26	FLEXSPI_EnableAHBParallelMode	869
34.6.27	FLEXSPI_UpdateLUT	869
34.6.28	FLEXSPI_WriteData	870
34.6.29	FLEXSPI_ReadData	870
34.6.30	FLEXSPI_WriteBlocking	870
34.6.31	FLEXSPI_ReadBlocking	871
34.6.32	FLEXSPI_TransferBlocking	872
34.6.33	FLEXSPI_TransferCreateHandle	872
34.6.34	FLEXSPI_TransferNonBlocking	873
34.6.35	FLEXSPI_TransferGetCount	873
34.6.36	FLEXSPI_TransferAbort	874
34.6.37	FLEXSPI_TransferHandleIRQ	874

Chapter 35 GPT: General Purpose Timer

35.1	Overview	875
35.2	Function groups	875
35.2.1	Initialization and deinitialization	875
35.3	Typical use case	875
35.3.1	GPT interrupt example	875
35.4	Data Structure Documentation	878
35.4.1	struct gpt_config_t	878
35.5	Enumeration Type Documentation	879
35.5.1	gpt_clock_source_t	879
35.5.2	gpt_input_capture_channel_t	879
35.5.3	gpt_input_operation_mode_t	880
35.5.4	gpt_output_compare_channel_t	880
35.5.5	gpt_output_operation_mode_t	880
35.5.6	gpt_interrupt_enable_t	880
35.5.7	gpt_status_flag_t	881

Section No.	Title	Page No.
35.6	Function Documentation	881
35.6.1	GPT_Init	881
35.6.2	GPT_Deinit	881
35.6.3	GPT_GetDefaultConfig	881
35.6.4	GPT_SoftwareReset	882
35.6.5	GPT_SetClockSource	882
35.6.6	GPT_GetClockSource	882
35.6.7	GPT_SetClockDivider	882
35.6.8	GPT_GetClockDivider	883
35.6.9	GPT_SetOscClockDivider	883
35.6.10	GPT_GetOscClockDivider	883
35.6.11	GPT_StartTimer	883
35.6.12	GPT_StopTimer	884
35.6.13	GPT_GetCurrentTimerCount	884
35.6.14	GPT_SetInputOperationMode	884
35.6.15	GPT_GetInputOperationMode	884
35.6.16	GPT_GetInputCaptureValue	885
35.6.17	GPT_SetOutputOperationMode	885
35.6.18	GPT_GetOutputOperationMode	886
35.6.19	GPT_SetOutputCompareValue	886
35.6.20	GPT_GetOutputCompareValue	886
35.6.21	GPT_ForceOutput	887
35.6.22	GPT_EnableInterrupts	887
35.6.23	GPT_DisableInterrupts	887
35.6.24	GPT_GetEnabledInterrupts	887
35.6.25	GPT_GetStatusFlags	888
35.6.26	GPT_ClearStatusFlags	888
Chapter 36	GPIO: General-Purpose Input/Output Driver	
36.1	Overview	889
36.2	Typical use case	889
36.2.1	Input Operation	889
36.3	Data Structure Documentation	891
36.3.1	struct gpio_pin_config_t	891
36.4	Macro Definition Documentation	891
36.4.1	FSL_GPIO_DRIVER_VERSION	891
36.5	Enumeration Type Documentation	891
36.5.1	gpio_pin_direction_t	891
36.5.2	gpio_interrupt_mode_t	891

Section No.	Title	Page No.
36.6	Function Documentation	892
36.6.1	GPIO_PinInit	892
36.6.2	GPIO_PinWrite	893
36.6.3	GPIO_WritePinOutput	893
36.6.4	GPIO_PortSet	893
36.6.5	GPIO_SetPinsOutput	893
36.6.6	GPIO_PortClear	894
36.6.7	GPIO_ClearPinsOutput	895
36.6.8	GPIO_PortToggle	895
36.6.9	GPIO_PinRead	895
36.6.10	GPIO_ReadPinInput	895
36.6.11	GPIO_PinReadPadStatus	896
36.6.12	GPIO_ReadPadStatus	897
36.6.13	GPIO_PinSetInterruptConfig	897
36.6.14	GPIO_SetPinInterruptConfig	897
36.6.15	GPIO_PortEnableInterrupts	897
36.6.16	GPIO_EnableInterrupts	898
36.6.17	GPIO_PortDisableInterrupts	898
36.6.18	GPIO_DisableInterrupts	898
36.6.19	GPIO_PortGetInterruptFlags	898
36.6.20	GPIO_GetPinsInterruptFlags	899
36.6.21	GPIO_PortClearInterruptFlags	899
36.6.22	GPIO_ClearPinsInterruptFlags	899
Chapter 37	KPP: KeyPad Port Driver	
37.1	Overview	901
37.2	Typical use case	901
37.3	Data Structure Documentation	902
37.3.1	struct kpp_config_t	902
37.4	Macro Definition Documentation	902
37.4.1	FSL_KPP_DRIVER_VERSION	902
37.5	Enumeration Type Documentation	903
37.5.1	kpp_interrupt_enable_t	903
37.5.2	kpp_sync_operation_t	903
37.6	Function Documentation	903
37.6.1	KPP_Init	903
37.6.2	KPP_Deinit	903
37.6.3	KPP_EnableInterrupts	903
37.6.4	KPP_DisableInterrupts	904

Section No.	Title	Page No.
37.6.5	KPP_GetStatusFlag	904
37.6.6	KPP_ClearStatusFlag	904
37.6.7	KPP_SetSynchronizeChain	905
37.6.8	KPP_keyPressScanning	906
 Chapter 38 LCDIFv2: LCD Interface v2		
38.1	Overview	907
38.2	Shadow load	907
38.3	Typical use case	907
38.4	Data Structure Documentation	912
38.4.1	struct lcdifv2_display_config_t	912
38.4.2	struct lcdifv2_buffer_config_t	913
38.4.3	struct lcdifv2_blend_config_t	913
38.5	Macro Definition Documentation	913
38.5.1	LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT	913
38.5.2	LCDIFV2_MAKE_DMA_DONE_INTERRUPT	914
38.5.3	LCDIFV2_MAKE_DMA_ERROR_INTERRUPT	914
38.6	Enumeration Type Documentation	914
38.6.1	_lcdifv2_polarity_flags	914
38.6.2	_lcdifv2_interrupt	914
38.6.3	lcdifv2_line_order_t	915
38.6.4	lcdifv2_csc_mode_t	915
38.6.5	lcdifv2_pixel_format_t	915
38.6.6	lcdifv2_alpha_mode_t	916
38.6.7	lcdifv2_pd_alpha_mode_t	916
38.6.8	lcdifv2_pd_color_mode_t	916
38.6.9	lcdifv2_pd_global_alpha_mode_t	916
38.6.10	lcdifv2_pd_factor_mode_t	917
38.6.11	lcdifv2_pd_blend_mode_t	917
38.6.12	lcdifv2_pd_layer_t	917
38.7	Function Documentation	917
38.7.1	LCDIFV2_Init	917
38.7.2	LCDIFV2_Deinit	918
38.7.3	LCDIFV2_Reset	918
38.7.4	LCDIFV2_DisplayGetDefaultConfig	918
38.7.5	LCDIFV2_SetDisplayConfig	919
38.7.6	LCDIFV2_EnableDisplay	919
38.7.7	LCDIFV2_EnableInterrupts	919
38.7.8	LCDIFV2_DisableInterrupts	919

Section No.	Title	Page No.
38.7.9	LCDIFV2_GetInterruptStatus	920
38.7.10	LCDIFV2_ClearInterruptStatus	920
38.7.11	LCDIFV2_SetLut	920
38.7.12	LCDIFV2_SetLayerSize	921
38.7.13	LCDIFV2_SetLayerOffset	921
38.7.14	LCDIFV2_SetLayerBufferConfig	922
38.7.15	LCDIFV2_SetLayerBufferAddr	922
38.7.16	LCDIFV2_EnableLayer	922
38.7.17	LCDIFV2_TriggerLayerShadowLoad	923
38.7.18	LCDIFV2_SetLayerBackgroundColor	923
38.7.19	LCDIFV2_SetLayerBlendConfig	923
38.7.20	LCDIFV2_SetCscMode	924
38.7.21	LCDIFV2_GetPorterDuffConfig	924

Chapter 39 LPADC: 12-bit SAR Analog-to-Digital Converter Driver

39.1	Overview	926
39.2	Typical use case	926
39.2.1	Polling Configuration	926
39.2.2	Interrupt Configuration	926
39.3	Data Structure Documentation	929
39.3.1	struct lpadc_config_t	929
39.3.2	struct lpadc_conv_command_config_t	930
39.3.3	struct lpadc_conv_trigger_config_t	932
39.3.4	struct lpadc_conv_result_t	932
39.4	Macro Definition Documentation	933
39.4.1	FSL_LPADC_DRIVER_VERSION	933
39.4.2	LPADC_GET_ACTIVE_COMMAND_STATUS	933
39.4.3	LPADC_GET_ACTIVE_TRIGGER_STATUE	933
39.5	Enumeration Type Documentation	933
39.5.1	_lpadc_status_flags	933
39.5.2	_lpadc_interrupt_enable	933
39.5.3	lpadc_sample_scale_mode_t	934
39.5.4	lpadc_sample_channel_mode_t	934
39.5.5	lpadc_hardware_average_mode_t	934
39.5.6	lpadc_sample_time_mode_t	935
39.5.7	lpadc_hardware_compare_mode_t	935
39.5.8	lpadc_reference_voltage_source_t	935
39.5.9	lpadc_power_level_mode_t	936
39.5.10	lpadc_trigger_priority_policy_t	936

Section No.	Title	Page No.
39.6	Function Documentation	936
39.6.1	LPADC_Init	936
39.6.2	LPADC_GetDefaultConfig	936
39.6.3	LPADC_Deinit	937
39.6.4	LPADC_Enable	937
39.6.5	LPADC_DoResetFIFO	937
39.6.6	LPADC_DoResetConfig	937
39.6.7	LPADC_GetStatusFlags	938
39.6.8	LPADC_ClearStatusFlags	938
39.6.9	LPADC_EnableInterrupts	938
39.6.10	LPADC_DisableInterrupts	938
39.6.11	LPADC_EnableFIFOWatermarkDMA	939
39.6.12	LPADC_GetConvResultCount	939
39.6.13	LPADC_GetConvResult	939
39.6.14	LPADC_SetConvTriggerConfig	940
39.6.15	LPADC_GetDefaultConvTriggerConfig	941
39.6.16	LPADC_DoSoftwareTrigger	941
39.6.17	LPADC_SetConvCommandConfig	941
39.6.18	LPADC_GetDefaultConvCommandConfig	942
Chapter 40	LPI2C: Low Power Inter-Integrated Circuit Driver	
40.1	Overview	943
40.2	Macro Definition Documentation	943
40.2.1	FSL_LPI2C_DRIVER_VERSION	943
40.2.2	I2C_RETRY_TIMES	944
40.3	Enumeration Type Documentation	944
40.3.1	anonymous enum	944
40.4	LPI2C Master Driver	945
40.4.1	Overview	945
40.4.2	Data Structure Documentation	948
40.4.3	Typedef Documentation	952
40.4.4	Enumeration Type Documentation	953
40.4.5	Function Documentation	955
40.5	LPI2C Slave Driver	969
40.5.1	Overview	969
40.5.2	Data Structure Documentation	971
40.5.3	Typedef Documentation	974
40.5.4	Enumeration Type Documentation	976
40.5.5	Function Documentation	977

Section No.	Title	Page No.
40.6	LPI2C Master DMA Driver	986
40.6.1	Overview	986
40.6.2	Data Structure Documentation	986
40.6.3	Typedef Documentation	987
40.6.4	Function Documentation	989
40.7	LPI2C FreeRTOS Driver	992
40.7.1	Overview	992
40.7.2	Macro Definition Documentation	992
40.7.3	Function Documentation	992
40.8	LPI2C CMSIS Driver	995
40.8.1	LPI2C CMSIS Driver	995
Chapter 41 LPSPI: Low Power Serial Peripheral Interface		
41.1	Overview	997
41.2	LPSPI Peripheral driver	998
41.2.1	Overview	998
41.2.2	Function groups	998
41.2.3	Typical use case	998
41.2.4	Data Structure Documentation	1005
41.2.5	Macro Definition Documentation	1011
41.2.6	Typedef Documentation	1011
41.2.7	Enumeration Type Documentation	1012
41.2.8	Function Documentation	1017
41.2.9	Variable Documentation	1032
41.3	LPSPI eDMA Driver	1033
41.3.1	Overview	1033
41.3.2	Data Structure Documentation	1034
41.3.3	Macro Definition Documentation	1038
41.3.4	Typedef Documentation	1038
41.3.5	Function Documentation	1038
41.4	LPSPI FreeRTOS Driver	1044
41.4.1	Overview	1044
41.4.2	Macro Definition Documentation	1044
41.4.3	Function Documentation	1044
41.5	LPSPI CMSIS Driver	1047
41.5.1	Function groups	1047
41.5.2	Typical use case	1048

Section No.	Title	Page No.
Chapter 42 LPUART: Low Power Universal Asynchronous Receiver/Transmitter Driver		
42.1	Overview	1049
42.2	LPUART Driver	1050
42.2.1	Overview	1050
42.2.2	Typical use case	1050
42.2.3	Data Structure Documentation	1055
42.2.4	Macro Definition Documentation	1058
42.2.5	Typedef Documentation	1058
42.2.6	Enumeration Type Documentation	1058
42.2.7	Function Documentation	1061
42.3	LPUART eDMA Driver	1078
42.3.1	Overview	1078
42.3.2	Data Structure Documentation	1079
42.3.3	Macro Definition Documentation	1079
42.3.4	Typedef Documentation	1079
42.3.5	Function Documentation	1080
42.4	LPUART FreeRTOS Driver	1084
42.4.1	Overview	1084
42.4.2	Data Structure Documentation	1084
42.4.3	Macro Definition Documentation	1085
42.4.4	Function Documentation	1085
42.5	LPUART CMSIS Driver	1087
42.5.1	Function groups	1087
Chapter 43 MECC: internal error correction code		
43.1	Overview	1089
43.2	Data Structure Documentation	1092
43.2.1	struct mecc_config_t	1092
43.2.2	struct mecc_single_error_info_t	1092
43.2.3	struct mecc_multi_error_info_t	1093
43.3	Macro Definition Documentation	1093
43.3.1	FSL_MECC_DRIVER_VERSION	1093
43.4	Enumeration Type Documentation	1093
43.4.1	anonymous enum	1093
43.4.2	anonymous enum	1093
43.4.3	anonymous enum	1094
43.4.4	anonymous enum	1095

Section No.	Title	Page No.
43.4.5	anonymous enum	1095
43.4.6	anonymous enum	1095
43.5	Function Documentation	1095
43.5.1	MECC_Init	1095
43.5.2	MECC_Deinit	1096
43.5.3	MECC_GetDefaultConfig	1096
43.5.4	MECC_GetStatusFlags	1096
43.5.5	MECC_ClearStatusFlags	1096
43.5.6	MECC_EnableInterruptStatus	1097
43.5.7	MECC_DisableInterruptStatus	1097
43.5.8	MECC_EnableInterrupts	1097
43.5.9	MECC_DisableInterrupts	1097
43.5.10	MECC_ErrorInjection	1098
43.5.11	MECC_GetSingleErrorInfo	1098
43.5.12	MECC_GetMultiErrorInfo	1099
Chapter 44	MIPI CSI2 RX: MIPI CSI2 RX Driver	
44.1	Overview	1100
44.2	Data Structure Documentation	1102
44.2.1	struct csi2rx_config_t	1102
44.3	Macro Definition Documentation	1103
44.3.1	FSL_CSI2RX_DRIVER_VERSION	1103
44.4	Enumeration Type Documentation	1103
44.4.1	_csi2rx_data_lane	1103
44.4.2	_csi2rx_payload	1103
44.4.3	_csi2rx_bit_error	1104
44.4.4	csi2rx_ppi_error_t	1104
44.4.5	_csi2rx_interrupt	1104
44.4.6	_csi2rx_ulps_status	1104
44.5	Function Documentation	1105
44.5.1	CSI2RX_Init	1105
44.5.2	CSI2RX_Deinit	1105
44.5.3	CSI2RX_GetBitError	1105
44.5.4	CSI2RX_GetEccBitErrorPosition	1106
44.5.5	CSI2RX_GetUlpsStatus	1106
44.5.6	CSI2RX_GetPpiErrorDataLanes	1107
44.5.7	CSI2RX_EnableInterrupts	1107
44.5.8	CSI2RX_DisableInterrupts	1108
44.5.9	CSI2RX_GetInterruptStatus	1108

Section No.	Title	Page No.
Chapter 45 MIPI_DSI: MIPI DSI Host Controller		
45.1	Overview	1109
45.2	MIPI DSI Driver	1110
45.2.1	Overview	1110
45.2.2	Command mode data transfer	1110
45.2.3	Data Structure Documentation	1118
45.2.4	Typedef Documentation	1123
45.2.5	Enumeration Type Documentation	1123
45.2.6	Function Documentation	1128
Chapter 46 MU: Messaging Unit		
46.1	Overview	1139
46.2	Function description	1139
46.2.1	MU initialization	1139
46.2.2	MU message	1139
46.2.3	MU flags	1140
46.2.4	Status and interrupt	1140
46.2.5	MU misc functions	1140
46.3	Macro Definition Documentation	1142
46.3.1	FSL_MU_DRIVER_VERSION	1142
46.4	Enumeration Type Documentation	1142
46.4.1	_mu_status_flags	1143
46.4.2	_mu_interrupt_enable	1143
46.4.3	_mu_interrupt_trigger	1143
46.5	Function Documentation	1144
46.5.1	MU_Init	1144
46.5.2	MU_Deinit	1144
46.5.3	MU_SendMsgNonBlocking	1144
46.5.4	MU_SendMsg	1145
46.5.5	MU_ReceiveMsgNonBlocking	1145
46.5.6	MU_ReceiveMsg	1145
46.5.7	MU_SetFlagsNonBlocking	1146
46.5.8	MU_SetFlags	1146
46.5.9	MU_GetFlags	1147
46.5.10	MU_GetStatusFlags	1147
46.5.11	MU_GetInterruptsPending	1148
46.5.12	MU_ClearStatusFlags	1149
46.5.13	MU_EnableInterrupts	1149
46.5.14	MU_DisableInterrupts	1150

Section No.	Title	Page No.
46.5.15	MU_TriggerInterrupts	1150
Chapter 47 OCOTP: On Chip One-Time Programmable controller.		
47.1	Overview	1152
47.2	OCOTP function group	1152
47.2.1	Initialization and de-initialization	1152
47.2.2	Read and Write operation	1152
47.3	OCOTP example	1152
47.4	Macro Definition Documentation	1153
47.4.1	FSL_OCOTP_DRIVER_VERSION	1153
47.5	Enumeration Type Documentation	1153
47.5.1	anonymous enum	1154
47.6	Function Documentation	1154
47.6.1	OCOTP_Init	1154
47.6.2	OCOTP_Deinit	1154
47.6.3	OCOTP_CheckBusyStatus	1154
47.6.4	OCOTP_CheckErrorStatus	1155
47.6.5	OCOTP_ClearErrorStatus	1155
47.6.6	OCOTP_ReloadShadowRegister	1155
47.6.7	OCOTP_ReadFuseShadowRegister	1156
47.6.8	OCOTP_ReadFuseShadowRegisterExt	1156
47.6.9	OCOTP_WriteFuseShadowRegister	1156
47.6.10	OCOTP_WriteFuseShadowRegisterWithLock	1157
47.6.11	OCOTP_GetVersion	1158
Chapter 48 PDM: Microphone Interface		
48.1	Overview	1159
48.2	PDM Driver	1160
48.2.1	Overview	1160
48.2.2	Typical use case	1160
48.2.3	Data Structure Documentation	1167
48.2.4	Enumeration Type Documentation	1170
48.2.5	Function Documentation	1175
48.3	PDM EDMA Driver	1192
48.3.1	Overview	1192
48.3.2	Data Structure Documentation	1193
48.3.3	Function Documentation	1194

Section No.	Title	Page No.
Chapter 49 PIT: Periodic Interrupt Timer		
49.1	Overview	1198
49.2	Function groups	1198
49.2.1	Initialization and deinitialization	1198
49.2.2	Timer period Operations	1198
49.2.3	Start and Stop timer operations	1198
49.2.4	Status	1199
49.2.5	Interrupt	1199
49.3	Typical use case	1199
49.3.1	PIT tick example	1199
49.4	Data Structure Documentation	1200
49.4.1	struct pit_config_t	1200
49.5	Enumeration Type Documentation	1201
49.5.1	pit_chnl_t	1201
49.5.2	pit_interrupt_enable_t	1201
49.5.3	pit_status_flags_t	1201
49.6	Function Documentation	1201
49.6.1	PIT_Init	1201
49.6.2	PIT_Deinit	1202
49.6.3	PIT_GetDefaultConfig	1202
49.6.4	PIT_SetTimerChainMode	1202
49.6.5	PIT_EnableInterrupts	1203
49.6.6	PIT_DisableInterrupts	1203
49.6.7	PIT_GetEnabledInterrupts	1203
49.6.8	PIT_GetStatusFlags	1204
49.6.9	PIT_ClearStatusFlags	1205
49.6.10	PIT_SetTimerPeriod	1205
49.6.11	PIT_GetCurrentTimerCount	1206
49.6.12	PIT_StartTimer	1206
49.6.13	PIT_StopTimer	1206
49.6.14	PIT_GetLifetimeTimerCount	1207
Chapter 50 PUF: Physical Unclonable Function		
50.1	Overview	1208
50.2	PUF Driver Initialization and deinitialization	1208
50.3	Comments about API usage in RTOS	1208

Section No.	Title	Page No.
50.4	Comments about API usage in interrupt handler	1208
50.5	PUF Driver Examples	1208
50.5.1	Simple examples	1208
50.6	Macro Definition Documentation	1209
50.6.1	FSL_PUF_DRIVER_VERSION	1209
50.6.2	PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE	1210
50.7	Enumeration Type Documentation	1210
50.7.1	puf_key_slot_t	1210
50.7.2	anonymous enum	1210
 Chapter 51 PWM: Pulse Width Modulator		
51.1	Overview	1211
51.2	PWM: Pulse Width Modulator	1211
51.2.1	Initialization and deinitialization	1211
51.2.2	PWM Operations	1211
51.2.3	Input capture operations	1211
51.2.4	Fault operation	1211
51.2.5	PWM Start and Stop operations	1212
51.2.6	Status	1212
51.2.7	Interrupt	1212
51.3	Register Update	1212
51.4	Typical use case	1212
51.4.1	PWM output	1212
51.5	Data Structure Documentation	1220
51.5.1	struct pwm_signal_param_t	1220
51.5.2	struct pwm_config_t	1220
51.5.3	struct pwm_fault_input_filter_param_t	1221
51.5.4	struct pwm_fault_param_t	1221
51.5.5	struct pwm_input_capture_param_t	1221
51.6	Enumeration Type Documentation	1222
51.6.1	pwm_submodule_t	1222
51.6.2	pwm_value_register_t	1222
51.6.3	_pwm_value_register_mask	1222
51.6.4	pwm_clock_source_t	1223
51.6.5	pwm_clock_prescale_t	1223
51.6.6	pwm_force_output_trigger_t	1223
51.6.7	pwm_init_source_t	1224

Section No.	Title	Page No.
51.6.8	pwm_load_frequency_t	1224
51.6.9	pwm_fault_input_t	1224
51.6.10	pwm_fault_disable_t	1225
51.6.11	pwm_input_capture_edge_t	1225
51.6.12	pwm_force_signal_t	1225
51.6.13	pwm_chnl_pair_operation_t	1225
51.6.14	pwm_register_reload_t	1226
51.6.15	pwm_fault_recovery_mode_t	1226
51.6.16	pwm_interrupt_enable_t	1226
51.6.17	pwm_status_flags_t	1227
51.6.18	pwm_dma_enable_t	1227
51.6.19	pwm_dma_source_select_t	1227
51.6.20	pwm_watermark_control_t	1228
51.6.21	pwm_mode_t	1228
51.6.22	pwm_level_select_t	1228
51.6.23	pwm_fault_state_t	1228
51.6.24	pwm_reload_source_select_t	1228
51.6.25	pwm_fault_clear_t	1229
51.6.26	pwm_module_control_t	1229
51.7	Function Documentation	1229
51.7.1	PWM_Init	1229
51.7.2	PWM_Deinit	1229
51.7.3	PWM_GetDefaultConfig	1230
51.7.4	PWM_SetupPwm	1230
51.7.5	PWM_UpdatePwmDutycycle	1231
51.7.6	PWM_UpdatePwmDutycycleHighAccuracy	1231
51.7.7	PWM_SetupInputCapture	1232
51.7.8	PWM_SetupFaultInputFilter	1232
51.7.9	PWM_SetupFaults	1232
51.7.10	PWM_FaultDefaultConfig	1233
51.7.11	PWM_SetupForceSignal	1233
51.7.12	PWM_EnableInterrupts	1233
51.7.13	PWM_DisableInterrupts	1234
51.7.14	PWM_GetEnabledInterrupts	1234
51.7.15	PWM_DMAFIFOWatermarkControl	1234
51.7.16	PWM_DMACaptureSourceSelect	1235
51.7.17	PWM_EnableDMACapture	1235
51.7.18	PWM_EnableDMAWrite	1235
51.7.19	PWM_GetStatusFlags	1236
51.7.20	PWM_ClearStatusFlags	1236
51.7.21	PWM_StartTimer	1236
51.7.22	PWM_StopTimer	1237
51.7.23	PWM_OutputTriggerEnable	1237
51.7.24	PWM_ActivateOutputTrigger	1237

Section No.	Title	Page No.
51.7.25	PWM_DeactivateOutputTrigger	1238
51.7.26	PWM_SetupSwCtrlOut	1238
51.7.27	PWM_SetPwmLdok	1239
51.7.28	PWM_SetPwmFaultState	1239
51.7.29	PWM_SetupFaultDisableMap	1239

Chapter 52 PXP: Pixel Pipeline

52.1	Overview	1241
52.2	Typical use case	1241
52.2.1	PXP normal operation	1241
52.2.2	PXP operation queue	1241
52.3	Data Structure Documentation	1248
52.3.1	struct pxp_output_buffer_config_t	1248
52.3.2	struct pxp_ps_buffer_config_t	1249
52.3.3	struct pxp_as_buffer_config_t	1250
52.3.4	struct pxp_as_blend_config_t	1250
52.3.5	struct pxp_csc2_config_t	1250
52.3.6	struct pxp_dither_final_lut_data_t	1252
52.3.7	struct pxp_dither_config_t	1252
52.3.8	struct pxp_porter_duff_config_t	1254
52.3.9	struct pxp_pic_copy_config_t	1255
52.4	Enumeration Type Documentation	1256
52.4.1	_pxp_interrupt_enable	1256
52.4.2	_pxp_flags	1256
52.4.3	pxp_flip_mode_t	1256
52.4.4	pxp_rotate_position_t	1257
52.4.5	pxp_rotate_degree_t	1257
52.4.6	pxp_interlaced_output_mode_t	1257
52.4.7	pxp_output_pixel_format_t	1257
52.4.8	pxp_ps_pixel_format_t	1258
52.4.9	pxp_ps_yuv_format_t	1258
52.4.10	pxp_as_pixel_format_t	1258
52.4.11	pxp_alpha_mode_t	1259
52.4.12	pxp_rop_mode_t	1259
52.4.13	pxp_block_size_t	1259
52.4.14	pxp_csc1_mode_t	1260
52.4.15	pxp_csc2_mode_t	1260
52.4.16	pxp_ram_t	1260
52.4.17	_pxp_dither_mode	1260
52.4.18	_pxp_dither_lut_mode	1260
52.4.19	_pxp_dither_matrix_size	1261

Section No.	Title	Page No.
52.4.20	anonymous enum	1261
52.4.21	anonymous enum	1261
52.4.22	anonymous enum	1261
52.4.23	anonymous enum	1261
52.4.24	pxp_porter_duff_blend_mode_t	1262
52.5	Function Documentation	1262
52.5.1	PXP_Init	1262
52.5.2	PXP_Deinit	1262
52.5.3	PXP_Reset	1263
52.5.4	PXP_Start	1263
52.5.5	PXP_EnableLcdHandShake	1263
52.5.6	PXP_EnableContinuousRun	1263
52.5.7	PXP_SetProcessBlockSize	1264
52.5.8	PXP_GetStatusFlags	1264
52.5.9	PXP_ClearStatusFlags	1264
52.5.10	PXP_GetAxiErrorId	1265
52.5.11	PXP_EnableInterrupts	1265
52.5.12	PXP_DisableInterrupts	1265
52.5.13	PXP_SetAlphaSurfaceBufferConfig	1266
52.5.14	PXP_SetAlphaSurfaceBlendConfig	1266
52.5.15	PXP_SetAlphaSurfaceOverlayColorKey	1266
52.5.16	PXP_EnableAlphaSurfaceOverlayColorKey	1267
52.5.17	PXP_SetAlphaSurfacePosition	1268
52.5.18	PXP_SetProcessSurfaceBackgroundColor	1268
52.5.19	PXP_SetProcessSurfaceBufferConfig	1268
52.5.20	PXP_SetProcessSurfaceScaler	1269
52.5.21	PXP_SetProcessSurfacePosition	1270
52.5.22	PXP_SetProcessSurfaceColorKey	1270
52.5.23	PXP_SetProcessSurfaceYUVFormat	1271
52.5.24	PXP_SetOutputBufferConfig	1271
52.5.25	PXP_SetOverwrittenAlphaValue	1271
52.5.26	PXP_EnableOverWrittenAlpha	1271
52.5.27	PXP_SetRotateConfig	1272
52.5.28	PXP_SetNextCommand	1272
52.5.29	PXP_IsNextCommandPending	1273
52.5.30	PXP_CancelNextCommand	1273
52.5.31	PXP_SetCsc1Mode	1274
52.5.32	PXP_EnableCsc1	1274
52.5.33	PXP_SetPorterDuffConfig	1274
52.5.34	PXP_GetPorterDuffConfig	1274
52.5.35	PXP_StartPictureCopy	1275
52.5.36	PXP_StartMemCopy	1276

Section No.	Title	Page No.
Chapter 53 QTMR: Quad Timer Driver		
53.1	Overview	1277
53.2	Data Structure Documentation	1280
53.2.1	struct qtmr_config_t	1280
53.3	Enumeration Type Documentation	1281
53.3.1	qtmr_primary_count_source_t	1281
53.3.2	qtmr_input_source_t	1281
53.3.3	qtmr_counting_mode_t	1282
53.3.4	qtmr_output_mode_t	1282
53.3.5	qtmr_input_capture_edge_t	1282
53.3.6	qtmr_preload_control_t	1283
53.3.7	qtmr_debug_action_t	1283
53.3.8	qtmr_interrupt_enable_t	1283
53.3.9	qtmr_status_flags_t	1283
53.3.10	qtmr_channel_selection_t	1283
53.3.11	qtmr_dma_enable_t	1284
53.4	Function Documentation	1284
53.4.1	QTMR_Init	1284
53.4.2	QTMR_Deinit	1284
53.4.3	QTMR_GetDefaultConfig	1284
53.4.4	QTMR_SetupPwm	1285
53.4.5	QTMR_SetupInputCapture	1285
53.4.6	QTMR_EnableInterrupts	1286
53.4.7	QTMR_DisableInterrupts	1286
53.4.8	QTMR_GetEnabledInterrupts	1286
53.4.9	QTMR_GetStatus	1287
53.4.10	QTMR_ClearStatusFlags	1287
53.4.11	QTMR_SetTimerPeriod	1287
53.4.12	QTMR_GetCurrentTimerCount	1288
53.4.13	QTMR_StartTimer	1288
53.4.14	QTMR_StopTimer	1289
53.4.15	QTMR_EnableDma	1289
53.4.16	QTMR_DisableDma	1289
Chapter 54 RTWDOG: 32-bit Watchdog Timer		
54.1	Overview	1291
54.2	Typical use case	1291
54.3	Data Structure Documentation	1293
54.3.1	struct rtwdog_work_mode_t	1293

Section No.	Title	Page No.
54.3.2	struct rtwdog_config_t	1293
54.4	Macro Definition Documentation	1294
54.4.1	FSL_RTWD OG_DRIVER_VERSION	1294
54.5	Enumeration Type Documentation	1294
54.5.1	rtwdog_clock_source_t	1294
54.5.2	rtwdog_clock_prescaler_t	1294
54.5.3	rtwdog_test_mode_t	1294
54.5.4	_rtwdog_interrupt_enable_t	1294
54.5.5	_rtwdog_status_flags_t	1295
54.6	Function Documentation	1295
54.6.1	RTWD OG_GetDefaultConfig	1295
54.6.2	RTWD OG_Init	1295
54.6.3	RTWD OG_Deinit	1296
54.6.4	RTWD OG_Enable	1296
54.6.5	RTWD OG_Disable	1296
54.6.6	RTWD OG_EnableInterrupts	1297
54.6.7	RTWD OG_DisableInterrupts	1297
54.6.8	RTWD OG_GetStatusFlags	1297
54.6.9	RTWD OG_EnableWindowMode	1298
54.6.10	RTWD OG_CountToMesec	1298
54.6.11	RTWD OG_ClearStatusFlags	1298
54.6.12	RTWD OG_SetTimeoutValue	1299
54.6.13	RTWD OG_SetWindowValue	1299
54.6.14	RTWD OG_Unlock	1299
54.6.15	RTWD OG_Refresh	1300
54.6.16	RTWD OG_GetCounterValue	1300
Chapter 55	RDC: Resource Domain Controller	
55.1	Overview	1301
55.2	Data Structure Documentation	1302
55.2.1	struct rdc_hardware_config_t	1303
55.2.2	struct rdc_domain_assignment_t	1303
55.2.3	struct rdc_periph_access_config_t	1303
55.2.4	struct rdc_mem_access_config_t	1304
55.2.5	struct rdc_mem_status_t	1305
55.3	Enumeration Type Documentation	1305
55.3.1	_rdc_interrupts	1305
55.3.2	_rdc_flags	1305
55.3.3	_rdc_access_policy	1305

Section No.	Title	Page No.
55.4	Function Documentation	1305
55.4.1	RDC_Init	1306
55.4.2	RDC_Deinit	1307
55.4.3	RDC_GetHardwareConfig	1307
55.4.4	RDC_EnableInterrupts	1307
55.4.5	RDC_DisableInterrupts	1307
55.4.6	RDC_GetInterruptStatus	1308
55.4.7	RDC_ClearInterruptStatus	1308
55.4.8	RDC_GetStatus	1308
55.4.9	RDC_ClearStatus	1308
55.4.10	RDC_SetMasterDomainAssignment	1309
55.4.11	RDC_GetDefaultMasterDomainAssignment	1309
55.4.12	RDC_LockMasterDomainAssignment	1309
55.4.13	RDC_SetPeriphAccessConfig	1310
55.4.14	RDC_GetDefaultPeriphAccessConfig	1310
55.4.15	RDC_LockPeriphAccessConfig	1310
55.4.16	RDC_GetPeriphAccessPolicy	1311
55.4.17	RDC_SetMemAccessConfig	1311
55.4.18	RDC_GetDefaultMemAccessConfig	1311
55.4.19	RDC_LockMemAccessConfig	1312
55.4.20	RDC_SetMemAccessValid	1312
55.4.21	RDC_GetMemViolationStatus	1312
55.4.22	RDC_ClearMemViolationFlag	1313
55.4.23	RDC_GetMemAccessPolicy	1313
55.4.24	RDC_GetCurrentMasterDomainId	1313

Chapter 56 RDC_SEMA42: Hardware Semaphores Driver

56.1	Overview	1314
56.2	Macro Definition Documentation	1315
56.2.1	RDC_SEMA42_GATE_NUM_RESET_ALL	1315
56.2.2	RDC_SEMA42_GATEn	1315
56.2.3	RDC_SEMA42_GATE_COUNT	1315
56.3	Function Documentation	1315
56.3.1	RDC_SEMA42_Init	1315
56.3.2	RDC_SEMA42_Deinit	1315
56.3.3	RDC_SEMA42_TryLock	1316
56.3.4	RDC_SEMA42_Lock	1316
56.3.5	RDC_SEMA42_Unlock	1317
56.3.6	RDC_SEMA42_GetLockMasterIndex	1317
56.3.7	RDC_SEMA42_GetLockDomainID	1317
56.3.8	RDC_SEMA42_ResetGate	1318
56.3.9	RDC_SEMA42_ResetAllGates	1319

Section No.	Title	Page No.
Chapter 57 SAI: Serial Audio Interface		
57.1	Overview	1320
57.2	Typical configurations	1320
57.3	Typical use case	1321
57.3.1	SAI Send/receive using an interrupt method	1321
57.3.2	SAI Send/receive using a DMA method	1321
57.4	SAI Driver	1322
57.4.1	Overview	1322
57.4.2	Data Structure Documentation	1330
57.4.3	Macro Definition Documentation	1333
57.4.4	Enumeration Type Documentation	1334
57.4.5	Function Documentation	1338
57.5	SAI EDMA Driver	1368
57.5.1	Overview	1368
57.5.2	Data Structure Documentation	1369
57.5.3	Function Documentation	1370
Chapter 58 SEMA4: Hardware Semaphores Driver		
58.1	Overview	1381
58.2	Macro Definition Documentation	1382
58.2.1	SEMA4_GATE_NUM_RESET_ALL	1382
58.3	Function Documentation	1382
58.3.1	SEMA4_Init	1382
58.3.2	SEMA4_Deinit	1382
58.3.3	SEMA4_TryLock	1382
58.3.4	SEMA4_Lock	1383
58.3.5	SEMA4_Unlock	1383
58.3.6	SEMA4_GetLockProc	1383
58.3.7	SEMA4_ResetGate	1384
58.3.8	SEMA4_ResetAllGates	1384
58.3.9	SEMA4_EnableGateNotifyInterrupt	1385
58.3.10	SEMA4_DisableGateNotifyInterrupt	1385
58.3.11	SEMA4_GetGateNotifyStatus	1385
58.3.12	SEMA4_ResetGateNotify	1386
58.3.13	SEMA4_ResetAllGateNotify	1386

Section No.	Title	Page No.
Chapter 59 SEMC: Smart External DRAM Controller Driver		
59.1	Overview	1388
59.2	SEMC: Smart External DRAM Controller Driver	1388
59.2.1	SEMC Initialization Operation	1388
59.2.2	SEMC Interrupt Operation	1388
59.2.3	SEMC Memory access Operation	1388
59.3	Typical use case	1388
59.4	Data Structure Documentation	1395
59.4.1	struct semc_sdram_config_t	1395
59.4.2	struct semc_nand_timing_config_t	1398
59.4.3	struct semc_nand_config_t	1399
59.4.4	struct semc_nor_config_t	1400
59.4.5	struct semc_sram_config_t	1402
59.4.6	struct semc_dbi_config_t	1404
59.4.7	struct semc_queuea_weight_struct_t	1405
59.4.8	union semc_queuea_weight_t	1406
59.4.9	struct semc_queueb_weight_struct_t	1406
59.4.10	union semc_queueb_weight_t	1407
59.4.11	struct semc_axi_queueweight_t	1407
59.4.12	struct semc_config_t	1407
59.5	Macro Definition Documentation	1408
59.5.1	FSL_SEMC_DRIVER_VERSION	1408
59.6	Enumeration Type Documentation	1408
59.6.1	anonymous enum	1408
59.6.2	semc_mem_type_t	1409
59.6.3	semc_waitready_polarity_t	1409
59.6.4	semc_sdram_cs_t	1409
59.6.5	semc_sram_cs_t	1409
59.6.6	semc_nand_access_type_t	1409
59.6.7	semc_interrupt_enable_t	1410
59.6.8	semc_ipcmd_datasize_t	1410
59.6.9	semc_refresh_time_t	1410
59.6.10	semc_caslatency_t	1410
59.6.11	semc_sdram_column_bit_num_t	1410
59.6.12	sem_sdram_burst_len_t	1411
59.6.13	semc_nand_column_bit_num_t	1411
59.6.14	sem_nand_burst_len_t	1411
59.6.15	semc_norsram_column_bit_num_t	1412
59.6.16	sem_norsram_burst_len_t	1412
59.6.17	semc_dbi_column_bit_num_t	1412

Section No.	Title	Page No.
59.6.18	sem_dbi_burst_len_t	1413
59.6.19	semc_iomux_pin	1413
59.6.20	semc_iomux_nora27_pin	1413
59.6.21	smec_port_size_t	1413
59.6.22	semc_addr_mode_t	1414
59.6.23	semc_dqs_mode_t	1414
59.6.24	semc_adv_polarity_t	1414
59.6.25	semc_sync_mode_t	1414
59.6.26	semc_adv_level_control_t	1414
59.6.27	semc_rdy_polarity_t	1415
59.6.28	semc_ipcmd_nand_addrmode_t	1415
59.6.29	semc_ipcmd_nand_cmdmode_t	1415
59.6.30	semc_nand_address_option_t	1415
59.6.31	semc_ipcmd_nor_dbi_t	1416
59.6.32	semc_ipcmd_sram_t	1416
59.6.33	semc_ipcmd_sdram_t	1416
59.7	Function Documentation	1416
59.7.1	SEMC_GetDefaultConfig	1416
59.7.2	SEMC_Init	1417
59.7.3	SEMC_Deinit	1417
59.7.4	SEMC_ConfigureSDRAM	1417
59.7.5	SEMC_ConfigureNAND	1418
59.7.6	SEMC_ConfigureNOR	1418
59.7.7	SEMC_ConfigureSRAMWithChipSelection	1418
59.7.8	SEMC_ConfigureSRAM	1419
59.7.9	SEMC_ConfigureDBI	1419
59.7.10	SEMC_EnableInterrupts	1419
59.7.11	SEMC_DisableInterrupts	1420
59.7.12	SEMC_GetStatusFlag	1420
59.7.13	SEMC_ClearStatusFlags	1420
59.7.14	SEMC_IsInIdle	1421
59.7.15	SEMC_SendIPCommand	1422
59.7.16	SEMC_BuildNandIPCommand	1422
59.7.17	SEMC_IsNandReady	1423
59.7.18	SEMC_IPCommandNandWrite	1424
59.7.19	SEMC_IPCommandNandRead	1424
59.7.20	SEMC_IPCommandNorWrite	1424
59.7.21	SEMC_IPCommandNorRead	1425
 Chapter 60 Smart Card		
60.1	Overview	1426
60.2	SmartCard Driver Initialization	1426

Section No.	Title	Page No.
60.3	SmartCard Call diagram	1426
60.4	Data Structure Documentation	1428
60.4.1	struct smartcard_card_params_t	1428
60.4.2	struct smartcard_timers_state_t	1429
60.4.3	struct smartcard_interface_config_t	1430
60.4.4	struct smartcard_xfer_t	1431
60.4.5	struct smartcard_context_t	1431
60.5	Enumeration Type Documentation	1433
60.5.1	anonymous enum	1433
60.5.2	smartcard_control_t	1433
60.5.3	smartcard_direction_t	1433
60.6	Smart Card PHY TDA8035 Driver	1434
60.7	Smart Card PHY EMVSIM Driver	1435
60.8	Smart Card PHY GPIO Driver	1436
60.9	Smart Card UART Driver	1437
60.10	Smart Card EMVSIM Driver	1438
60.10.1	Overview	1438
60.10.2	Enumeration Type Documentation	1439
60.10.3	Function Documentation	1440
60.11	Smart Card PHY Driver	1444
60.11.1	Overview	1444
60.11.2	Macro Definition Documentation	1444
60.11.3	Function Documentation	1444
Chapter 61 SNVS: Secure Non-Volatile Storage		
61.1	Overview	1448
61.2	Secure Non-Volatile Storage High-Power	1449
61.2.1	Overview	1449
61.2.2	Data Structure Documentation	1452
61.2.3	Macro Definition Documentation	1453
61.2.4	Enumeration Type Documentation	1453
61.2.5	Function Documentation	1455
61.3	Secure Non-Volatile Storage Low-Power	1467
61.3.1	Overview	1467
61.3.2	Data Structure Documentation	1471

Section No.	Title	Page No.
61.3.3	Enumeration Type Documentation	1472
61.3.4	Function Documentation	1472
Chapter 62 SPDIF: Sony/Philips Digital Interface		
62.1	Overview	1484
62.2	Typical use case	1484
62.2.1	SPDIF Send/receive using an interrupt method	1484
62.2.2	SPDIF Send/receive using a DMA method	1484
62.3	Data Structure Documentation	1489
62.3.1	struct spdif_config_t	1489
62.3.2	struct spdif_transfer_t	1490
62.3.3	struct _spdif_handle	1490
62.4	Macro Definition Documentation	1490
62.4.1	SPDIF_XFER_QUEUE_SIZE	1490
62.5	Enumeration Type Documentation	1490
62.5.1	anonymous enum	1491
62.5.2	spdif_rxfull_select_t	1491
62.5.3	spdif_txempty_select_t	1491
62.5.4	spdif_uchannel_source_t	1492
62.5.5	spdif_gain_select_t	1492
62.5.6	spdif_tx_source_t	1492
62.5.7	spdif_validity_config_t	1492
62.5.8	anonymous enum	1492
62.5.9	anonymous enum	1493
62.6	Function Documentation	1493
62.6.1	SPDIF_Init	1493
62.6.2	SPDIF_GetDefaultConfig	1494
62.6.3	SPDIF_Deinit	1494
62.6.4	SPDIF_GetInstance	1494
62.6.5	SPDIF_TxFIFOReset	1494
62.6.6	SPDIF_RxFIFOReset	1495
62.6.7	SPDIF_TxEnable	1495
62.6.8	SPDIF_RxEnable	1495
62.6.9	SPDIF_GetStatusFlag	1495
62.6.10	SPDIF_ClearStatusFlags	1496
62.6.11	SPDIF_EnableInterrupts	1496
62.6.12	SPDIF_DisableInterrupts	1496
62.6.13	SPDIF_EnableDMA	1497
62.6.14	SPDIF_TxGetLeftDataRegisterAddress	1497

Section No.	Title	Page No.
62.6.15	SPDIF_TxGetRightDataRegisterAddress	1498
62.6.16	SPDIF_RxGetLeftDataRegisterAddress	1499
62.6.17	SPDIF_RxGetRightDataRegisterAddress	1499
62.6.18	SPDIF_TxSetSampleRate	1499
62.6.19	SPDIF_GetRxSampleRate	1500
62.6.20	SPDIF_WriteBlocking	1500
62.6.21	SPDIF_WriteLeftData	1500
62.6.22	SPDIF_WriteRightData	1501
62.6.23	SPDIF_WriteChannelStatusHigh	1501
62.6.24	SPDIF_WriteChannelStatusLow	1501
62.6.25	SPDIF_ReadBlocking	1501
62.6.26	SPDIF_ReadLeftData	1502
62.6.27	SPDIF_ReadRightData	1502
62.6.28	SPDIF_ReadChannelStatusHigh	1502
62.6.29	SPDIF_ReadChannelStatusLow	1503
62.6.30	SPDIF_ReadQChannel	1504
62.6.31	SPDIF_ReadUChannel	1504
62.6.32	SPDIF_TransferTxCreateHandle	1504
62.6.33	SPDIF_TransferRxCreateHandle	1505
62.6.34	SPDIF_TransferSendNonBlocking	1505
62.6.35	SPDIF_TransferReceiveNonBlocking	1506
62.6.36	SPDIF_TransferGetSendCount	1506
62.6.37	SPDIF_TransferGetReceiveCount	1507
62.6.38	SPDIF_TransferAbortSend	1507
62.6.39	SPDIF_TransferAbortReceive	1507
62.6.40	SPDIF_TransferTxHandleIRQ	1508
62.6.41	SPDIF_TransferRxHandleIRQ	1508
62.7	SPDIF eDMA Driver	1509
62.7.1	Overview	1509
62.7.2	Data Structure Documentation	1510
62.7.3	Function Documentation	1511
 Chapter 63 SSARC: State Save and Restore Controller		
63.1	Overview	1516
63.2	Data Structure Documentation	1518
63.2.1	struct ssarc_descriptor_config_t	1518
63.2.2	struct ssarc_group_config_t	1519
63.3	Macro Definition Documentation	1520
63.3.1	FSL_SSARC_DRIVER_VERSION	1520
63.4	Enumeration Type Documentation	1520

Section No.	Title	Page No.
63.4.1	<code>_ssarc_interrupt_status_flags</code>	1520
63.4.2	<code>ssarc_descriptor_register_size_t</code>	1520
63.4.3	<code>ssarc_descriptor_operation_t</code>	1520
63.4.4	<code>ssarc_descriptor_type_t</code>	1521
63.4.5	<code>ssarc_save_restore_order_t</code>	1521
63.4.6	<code>ssarc_software_trigger_mode_t</code>	1521
63.5	Function Documentation	1521
63.5.1	<code>SSARC_GetDescriptorRegisterAddress</code>	1521
63.5.2	<code>SSARC_GetDescriptorRegisterData</code>	1522
63.5.3	<code>SSARC_SetDescriptorConfig</code>	1522
63.5.4	<code>SSARC_GroupInit</code>	1522
63.5.5	<code>SSARC_GroupDeinit</code>	1523
63.5.6	<code>SSARC_LockGroupDomain</code>	1523
63.5.7	<code>SSARC_LockGroupWrite</code>	1523
63.5.8	<code>SSARC_LockGroupRead</code>	1524
63.5.9	<code>SSARC_TriggerSoftwareRequest</code>	1524
63.5.10	<code>SSARC_ResetWholeBlock</code>	1524
63.5.11	<code>SSARC_EnableHardwareRequest</code>	1525
63.5.12	<code>SSARC_GetStatusFlags</code>	1525
63.5.13	<code>SSARC_ClearStatusFlags</code>	1525
63.5.14	<code>SSARC_GetErrorIndex</code>	1526
63.5.15	<code>SSARC_SetTimeoutValue</code>	1526
63.5.16	<code>SSARC_GetTimeoutValue</code>	1526
63.5.17	<code>SSARC_GetHardwareRequestRestorePendingGroup</code>	1527
63.5.18	<code>SSARC_GetHardwareRequestSavePendingGroup</code>	1528
63.5.19	<code>SSARC_GetSoftwareRequestRestorePendingGroup</code>	1528
63.5.20	<code>SSARC_GetSoftwareRequestSavePendingGroup</code>	1528
Chapter 64	TEMPSENSOR: Temperature Sensor Module	
64.1	Overview	1530
64.2	TEMPSENSOR Operations	1530
64.3	Data Structure Documentation	1531
64.3.1	<code>struct tmpsns_config_t</code>	1531
64.4	Enumeration Type Documentation	1532
64.4.1	anonymous enum	1532
64.4.2	<code>tmpsns_measure_mode_t</code>	1532
64.4.3	<code>tmpsns_alarm_mode_t</code>	1532
64.5	Function Documentation	1532
64.5.1	<code>TMPSNS_Init</code>	1532

Section No.	Title	Page No.
64.5.2	TMPSNS_Deinit	1533
64.5.3	TMPSNS_GetDefaultConfig	1533
64.5.4	TMPSNS_StartMeasure	1533
64.5.5	TMPSNS_StopMeasure	1533
64.5.6	TMPSNS_GetCurrentTemperature	1533
64.5.7	TMPSNS_SetTempAlarm	1534
64.5.8	TMPSNS_EnableInterrupt	1534
64.5.9	TMPSNS_DisableInterrupt	1534

Chapter 65 USDHC: Ultra Secured Digital Host Controller Driver

65.1	Overview	1535
65.2	Typical use case	1535
65.2.1	USDHC Operation	1535
65.3	Data Structure Documentation	1546
65.3.1	struct usdhc_adma2_descriptor_t	1546
65.3.2	struct usdhc_capability_t	1546
65.3.3	struct usdhc_boot_config_t	1547
65.3.4	struct usdhc_config_t	1548
65.3.5	struct usdhc_command_t	1548
65.3.6	struct usdhc_adma_config_t	1549
65.3.7	struct usdhc_scatter_gather_data_list_t	1549
65.3.8	struct usdhc_scatter_gather_data_t	1549
65.3.9	struct usdhc_scatter_gather_transfer_t	1550
65.3.10	struct usdhc_data_t	1550
65.3.11	struct usdhc_transfer_t	1551
65.3.12	struct usdhc_transfer_callback_t	1552
65.3.13	struct _usdhc_handle	1552
65.3.14	struct usdhc_host_t	1553
65.4	Macro Definition Documentation	1553
65.4.1	FSL_USDHC_DRIVER_VERSION	1553
65.4.2	USDHC_ADMA1_ADDRESS_ALIGN	1554
65.4.3	USDHC_ADMA1_LENGTH_ALIGN	1554
65.4.4	USDHC_ADMA2_ADDRESS_ALIGN	1554
65.4.5	USDHC_ADMA2_LENGTH_ALIGN	1554
65.4.6	USDHC_ADMA1_DESCRIPTOR_ADDRESS_SHIFT	1554
65.4.7	USDHC_ADMA1_DESCRIPTOR_ADDRESS_MASK	1554
65.4.8	USDHC_ADMA1_DESCRIPTOR_LENGTH_SHIFT	1554
65.4.9	USDHC_ADMA1_DESCRIPTOR_LENGTH_MASK	1554
65.4.10	USDHC_ADMA1_DESCRIPTOR_MAX_LENGTH_PER_ENTRY	1555
65.4.11	USDHC_ADMA2_DESCRIPTOR_LENGTH_SHIFT	1555
65.4.12	USDHC_ADMA2_DESCRIPTOR_LENGTH_MASK	1555

Section No.	Title	Page No.
65.4.13	USDHC_ADMA2_DESCRIPTOR_MAX_LENGTH_PER_ENTRY	1555
65.5	Typedef Documentation	1555
65.5.1	usdhc_adma1_descriptor_t	1556
65.5.2	usdhc_transfer_function_t	1556
65.6	Enumeration Type Documentation	1556
65.6.1	anonymous enum	1556
65.6.2	anonymous enum	1556
65.6.3	anonymous enum	1557
65.6.4	anonymous enum	1557
65.6.5	anonymous enum	1557
65.6.6	anonymous enum	1558
65.6.7	anonymous enum	1558
65.6.8	anonymous enum	1559
65.6.9	anonymous enum	1559
65.6.10	anonymous enum	1560
65.6.11	anonymous enum	1560
65.6.12	anonymous enum	1560
65.6.13	usdhc_transfer_direction_t	1561
65.6.14	usdhc_data_bus_width_t	1561
65.6.15	usdhc_endian_mode_t	1561
65.6.16	usdhc_dma_mode_t	1561
65.6.17	anonymous enum	1562
65.6.18	usdhc_boot_mode_t	1562
65.6.19	usdhc_card_command_type_t	1562
65.6.20	usdhc_card_response_type_t	1562
65.6.21	anonymous enum	1563
65.6.22	anonymous enum	1563
65.6.23	anonymous enum	1563
65.6.24	usdhc_burst_len_t	1564
65.6.25	anonymous enum	1564
65.7	Function Documentation	1564
65.7.1	USDHC_Init	1564
65.7.2	USDHC_Deinit	1565
65.7.3	USDHC_Reset	1565
65.7.4	USDHC_SetAdmaTableConfig	1565
65.7.5	USDHC_SetInternalDmaConfig	1566
65.7.6	USDHC_SetADMA2Descriptor	1566
65.7.7	USDHC_SetADMA1Descriptor	1567
65.7.8	USDHC_EnableInternalDMA	1567
65.7.9	USDHC_EnableInterruptStatus	1568
65.7.10	USDHC_DisableInterruptStatus	1568
65.7.11	USDHC_EnableInterruptSignal	1568

Section No.	Title	Page No.
65.7.12	USDHC_DisableInterruptSignal	1568
65.7.13	USDHC_GetEnabledInterruptStatusFlags	1569
65.7.14	USDHC_GetInterruptStatusFlags	1569
65.7.15	USDHC_ClearInterruptStatusFlags	1569
65.7.16	USDHC_GetAutoCommand12ErrorStatusFlags	1570
65.7.17	USDHC_GetAdmaErrorStatusFlags	1571
65.7.18	USDHC_GetPresentStatusFlags	1571
65.7.19	USDHC_GetCapability	1571
65.7.20	USDHC_ForceClockOn	1572
65.7.21	USDHC_SetSdClock	1572
65.7.22	USDHC_SetCardActive	1572
65.7.23	USDHC_AssertHardwareReset	1573
65.7.24	USDHC_SetDataBusWidth	1573
65.7.25	USDHC_WriteData	1573
65.7.26	USDHC_ReadData	1573
65.7.27	USDHC_SendCommand	1574
65.7.28	USDHC_EnableWakeupEvent	1574
65.7.29	USDHC_CardDetectByData3	1574
65.7.30	USDHC_DetectCardInsert	1574
65.7.31	USDHC_EnableSdioControl	1575
65.7.32	USDHC_SetContinueRequest	1575
65.7.33	USDHC_RequestStopAtBlockGap	1575
65.7.34	USDHC_SetMmcBootConfig	1575
65.7.35	USDHC_EnableMmcBoot	1576
65.7.36	USDHC_SetForceEvent	1576
65.7.37	USDHC_SelectVoltage	1576
65.7.38	USDHC_RequestTuningForSDR50	1576
65.7.39	USDHC_RequestReTuning	1577
65.7.40	USDHC_EnableAutoTuning	1577
65.7.41	USDHC_EnableAutoTuningForCmdAndData	1577
65.7.42	USDHC_EnableManualTuning	1577
65.7.43	USDHC_GetTuningDelayStatus	1578
65.7.44	USDHC_SetTuningDelay	1578
65.7.45	USDHC_AdjustDelayForManualTuning	1578
65.7.46	USDHC_SetStandardTuningCounter	1579
65.7.47	USDHC_EnableStandardTuning	1579
65.7.48	USDHC_GetExecuteStdTuningStatus	1580
65.7.49	USDHC_CheckStdTuningResult	1581
65.7.50	USDHC_CheckTuningError	1581
65.7.51	USDHC_EnableDDRMode	1581
65.7.52	USDHC_EnableHS400Mode	1581
65.7.53	USDHC_ResetStrobeDLL	1582
65.7.54	USDHC_EnableStrobeDLL	1582
65.7.55	USDHC_ConfigStrobeDLL	1582
65.7.56	USDHC_SetStrobeDllOverride	1582

Section No.	Title	Page No.
65.7.57	USDHC_GetStrobeDLLStatus	1583
65.7.58	USDHC_SetDataConfig	1583
65.7.59	USDHC_TransferCreateHandle	1583
65.7.60	USDHC_TransferNonBlocking	1584
65.7.61	USDHC_TransferBlocking	1584
65.7.62	USDHC_TransferHandleIRQ	1585

Chapter 66 WDOG: Watchdog Timer Driver

66.1	Overview	1586
66.2	Typical use case	1586
66.3	Data Structure Documentation	1587
66.3.1	struct wdog_work_mode_t	1587
66.3.2	struct wdog_config_t	1587
66.4	Enumeration Type Documentation	1588
66.4.1	_wdog_interrupt_enable	1588
66.4.2	_wdog_status_flags	1588
66.5	Function Documentation	1588
66.5.1	WDOG_GetDefaultConfig	1588
66.5.2	WDOG_Init	1589
66.5.3	WDOG_Deinit	1589
66.5.4	WDOG_Enable	1589
66.5.5	WDOG_Disable	1590
66.5.6	WDOG_TriggerSystemSoftwareReset	1590
66.5.7	WDOG_TriggerSoftwareSignal	1590
66.5.8	WDOG_EnableInterrupts	1591
66.5.9	WDOG_GetStatusFlags	1592
66.5.10	WDOG_ClearInterruptStatus	1592
66.5.11	WDOG_SetTimeoutValue	1593
66.5.12	WDOG_SetInterruptTimeoutValue	1593
66.5.13	WDOG_DisablePowerDownEnable	1593
66.5.14	WDOG_Refresh	1594

Chapter 67 XECC: external error correction code controller

67.1	Overview	1595
67.2	Data Structure Documentation	1596
67.2.1	struct xecc_config_t	1596
67.2.2	struct xecc_single_error_info_t	1598
67.2.3	struct xecc_multi_error_info_t	1598

Section No.	Title	Page No.
67.3	Macro Definition Documentation	1598
67.3.1	FSL_XECC_DRIVER_VERSION	1598
67.4	Enumeration Type Documentation	1598
67.4.1	anonymous enum	1598
67.4.2	anonymous enum	1599
67.4.3	anonymous enum	1599
67.5	Function Documentation	1599
67.5.1	XECC_Init	1599
67.5.2	XECC_Deinit	1599
67.5.3	XECC_GetDefaultConfig	1599
67.5.4	XECC_GetStatusFlags	1600
67.5.5	XECC_ClearStatusFlags	1600
67.5.6	XECC_EnableInterruptStatus	1600
67.5.7	XECC_DisableInterruptStatus	1600
67.5.8	XECC_EnableInterrupts	1601
67.5.9	XECC_DisableInterrupts	1601
67.5.10	XECC_WriteECCEnable	1601
67.5.11	XECC_ReadECCEnable	1601
67.5.12	XECC_SwapECCEnable	1602
67.5.13	XECC_ErrorInjection	1602
67.5.14	XECC_GetSingleErrorInfo	1602
67.5.15	XECC_GetMultiErrorInfo	1603
Chapter 68	XBARA: Inter-Peripheral Crossbar Switch	
68.1	Overview	1604
68.2	Function	1604
68.2.1	XBARA Initialization	1604
68.2.2	Call diagram	1604
68.3	Typical use case	1604
68.4	Data Structure Documentation	1605
68.4.1	struct xbara_control_config_t	1605
68.5	Enumeration Type Documentation	1606
68.5.1	xbara_active_edge_t	1606
68.5.2	xbara_request_t	1606
68.5.3	xbara_status_flag_t	1606
68.6	Function Documentation	1607
68.6.1	XBARA_Init	1607
68.6.2	XBARA_Deinit	1608

Section No.	Title	Page No.
68.6.3	XBARA_SetSignalsConnection	1608
68.6.4	XBARA_GetStatusFlags	1608
68.6.5	XBARA_ClearStatusFlags	1609
68.6.6	XBARA_SetOutputSignalConfig	1609
 Chapter 69 XBARB: Inter-Peripheral Crossbar Switch		
69.1	Overview	1610
69.2	Function groups	1610
69.2.1	XBARB Initialization	1610
69.2.2	Call diagram	1610
69.3	Typical use case	1610
69.4	Function Documentation	1611
69.4.1	XBARB_Init	1611
69.4.2	XBARB_Deinit	1612
69.4.3	XBARB_SetSignalsConnection	1612
 Chapter 70 XRDC2: Extended Resource Domain Controller 2		
70.1	Overview	1613
70.2	XRDC2 functions	1613
70.3	Typical use case	1613
70.4	Data Structure Documentation	1617
70.4.1	struct xrdc2_master_domain_assignment_t	1617
70.4.2	struct xrdc2_periph_access_config_t	1618
70.4.3	struct xrdc2_mem_access_config_t	1618
70.4.4	struct xrdc2_mem_slot_access_config_t	1618
70.5	Macro Definition Documentation	1619
70.5.1	FSL_XRDC2_DRIVER_VERSION	1619
70.6	Enumeration Type Documentation	1619
70.6.1	xrdc2_global_config_lock_t	1619
70.6.2	xrdc2_secure_attr_t	1619
70.6.3	xrdc2_privilege_attr_t	1619
70.6.4	xrdc2_access_config_lock_t	1619
70.7	Function Documentation	1620
70.7.1	XRDC2_Init	1620
70.7.2	XRDC2_Deinit	1620

Section No.	Title	Page No.
70.7.3	XRDC2_SetGlobalValid	1620
70.7.4	XRDC2_GetCurrentMasterDomainId	1620
70.7.5	XRDC2_SetGlobalConfigLock	1621
70.7.6	XRDC2_GetCurrentGlobalConfigLockOwnerDomainId	1621
70.7.7	XRDC2_GetDefaultMasterDomainAssignment	1621
70.7.8	XRDC2_SetMasterDomainAssignment	1622
70.7.9	XRDC2_LockMasterDomainAssignment	1622
70.7.10	XRDC2_SetMasterDomainAssignmentValid	1622
70.7.11	XRDC2_GetMemSlotAccessDefaultConfig	1623
70.7.12	XRDC2_SetMemSlotAccessConfig	1623
70.7.13	XRDC2_SetMemSlotAccessValid	1623
70.7.14	XRDC2_SetMemSlotAccessLockMode	1624
70.7.15	XRDC2_SetMemSlotDomainAccessPolicy	1624
70.7.16	XRDC2_EnableMemSlotExclAccessLock	1624
70.7.17	XRDC2_GetMemSlotExclAccessLockDomainOwner	1625
70.7.18	XRDC2_TryLockMemSlotExclAccess	1625
70.7.19	XRDC2_LockMemSlotExclAccess	1625
70.7.20	XRDC2_UnlockMemSlotExclAccess	1626
70.7.21	XRDC2_ForceMemSlotExclAccessLockRelease	1626
70.7.22	XRDC2_GetMemAccessDefaultConfig	1626
70.7.23	XRDC2_SetMemAccessConfig	1627
70.7.24	XRDC2_SetMemAccessValid	1627
70.7.25	XRDC2_SetMemAccessLockMode	1627
70.7.26	XRDC2_SetMemDomainAccessPolicy	1627
70.7.27	XRDC2_EnableMemExclAccessLock	1628
70.7.28	XRDC2_GetMemExclAccessLockDomainOwner	1628
70.7.29	XRDC2_TryLockMemExclAccess	1628
70.7.30	XRDC2_LockMemExclAccess	1629
70.7.31	XRDC2_UnlockMemExclAccess	1629
70.7.32	XRDC2_ForceMemExclAccessLockRelease	1629
70.7.33	XRDC2_GetPeriphAccessDefaultConfig	1630
70.7.34	XRDC2_SetPeriphAccessConfig	1630
70.7.35	XRDC2_SetPeriphAccessValid	1630
70.7.36	XRDC2_SetPeriphAccessLockMode	1631
70.7.37	XRDC2_SetPeriphDomainAccessPolicy	1631
70.7.38	XRDC2_EnablePeriphExclAccessLock	1631
70.7.39	XRDC2_GetPeriphExclAccessLockDomainOwner	1632
70.7.40	XRDC2_TryLockPeriphExclAccess	1632
70.7.41	XRDC2_LockPeriphExclAccess	1632
70.7.42	XRDC2_UnlockPeriphExclAccess	1633
70.7.43	XRDC2_ForcePeriphExclAccessLockRelease	1633

Section No.	Title	Page No.
Chapter 71 Debug Console		
71.1	Overview	1634
71.2	Function groups	1634
71.2.1	Initialization	1634
71.2.2	Advanced Feature	1635
71.2.3	SDK_DEBUGCONSOLE and SDK_DEBUGCONSOLE_UART	1639
71.3	Typical use case	1640
71.4	Macro Definition Documentation	1642
71.4.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	1642
71.4.2	DEBUGCONSOLE_REDIRECT_TO_SDK	1642
71.4.3	DEBUGCONSOLE_DISABLE	1642
71.4.4	SDK_DEBUGCONSOLE	1642
71.4.5	PRINTF	1642
71.5	Function Documentation	1642
71.5.1	DbgConsole_Init	1642
71.5.2	DbgConsole_Deinit	1643
71.5.3	DbgConsole_EnterLowpower	1643
71.5.4	DbgConsole_ExitLowpower	1644
71.5.5	DbgConsole_Printf	1644
71.5.6	DbgConsole_Vprintf	1644
71.5.7	DbgConsole_Putchar	1644
71.5.8	DbgConsole_Scanf	1645
71.5.9	DbgConsole_Getchar	1645
71.5.10	DbgConsole_BlockingPrintf	1646
71.5.11	DbgConsole_BlockingVprintf	1646
71.5.12	DbgConsole_Flush	1646
71.5.13	StrFormatPrintf	1647
71.5.14	StrFormatScanf	1647
71.6	Semihosting	1648
71.6.1	Guide Semihosting for IAR	1648
71.6.2	Guide Semihosting for Keil μ Vision	1648
71.6.3	Guide Semihosting for MCUXpresso IDE	1649
71.6.4	Guide Semihosting for ARMGCC	1649
71.7	SWO	1652
71.7.1	Guide SWO for SDK	1652
71.7.2	Guide SWO for Keil μ Vision	1653
71.7.3	Guide SWO for MCUXpresso IDE	1654
71.7.4	Guide SWO for ARMGCC	1654

Section No.	Title	Page No.
Chapter 72 Notification Framework		
72.1	Overview	1655
72.2	Notifier Overview	1655
72.3	Data Structure Documentation	1657
72.3.1	struct notifier_notification_block_t	1657
72.3.2	struct notifier_callback_config_t	1658
72.3.3	struct notifier_handle_t	1658
72.4	Typedef Documentation	1659
72.4.1	notifier_user_config_t	1659
72.4.2	notifier_user_function_t	1659
72.4.3	notifier_callback_t	1660
72.5	Enumeration Type Documentation	1660
72.5.1	_notifier_status	1660
72.5.2	notifier_policy_t	1661
72.5.3	notifier_notification_type_t	1661
72.5.4	notifier_callback_type_t	1661
72.6	Function Documentation	1661
72.6.1	NOTIFIER_CreateHandle	1662
72.6.2	NOTIFIER_SwitchConfig	1663
72.6.3	NOTIFIER_GetErrorCallbackIndex	1664
Chapter 73 Shell		
73.1	Overview	1665
73.2	Function groups	1665
73.2.1	Initialization	1665
73.2.2	Advanced Feature	1665
73.2.3	Shell Operation	1665
73.3	Data Structure Documentation	1667
73.3.1	struct shell_command_t	1667
73.4	Macro Definition Documentation	1668
73.4.1	SHELL_NON_BLOCKING_MODE	1668
73.4.2	SHELL_AUTO_COMPLETE	1668
73.4.3	SHELL_BUFFER_SIZE	1668
73.4.4	SHELL_MAX_ARGS	1668
73.4.5	SHELL_HISTORY_COUNT	1668
73.4.6	SHELL_HANDLE_SIZE	1668

Section No.	Title	Page No.
73.4.7	SHELL_USE_COMMON_TASK	1668
73.4.8	SHELL_TASK_PRIORITY	1668
73.4.9	SHELL_TASK_STACK_SIZE	1668
73.4.10	SHELL_HANDLE_DEFINE	1669
73.4.11	SHELL_COMMAND_DEFINE	1669
73.4.12	SHELL_COMMAND	1670
73.5	Typedef Documentation	1670
73.5.1	cmd_function_t	1670
73.6	Enumeration Type Documentation	1670
73.6.1	shell_status_t	1670
73.7	Function Documentation	1670
73.7.1	SHELL_Init	1670
73.7.2	SHELL_RegisterCommand	1671
73.7.3	SHELL_UnregisterCommand	1672
73.7.4	SHELL_Write	1672
73.7.5	SHELL_Printf	1672
73.7.6	SHELL_WriteSynchronization	1673
73.7.7	SHELL_PrintfSynchronization	1673
73.7.8	SHELL_ChangePrompt	1674
73.7.9	SHELL_PrintPrompt	1674
73.7.10	SHELL_Task	1674
73.7.11	SHELL_checkRunningInIsr	1675
 Chapter 74 Cards: Secure Digital Card/Embedded MultiMedia Card/SDIO Card		
74.1	Overview	1676
74.2	SDIO Card Driver	1677
74.2.1	Overview	1677
74.2.2	SDIO CARD Operation	1677
74.2.3	Data Structure Documentation	1679
74.2.4	Macro Definition Documentation	1681
74.2.5	Enumeration Type Documentation	1681
74.2.6	Function Documentation	1681
74.3	SD Card Driver	1698
74.3.1	Overview	1698
74.3.2	SD CARD Operation	1698
74.3.3	Data Structure Documentation	1701
74.3.4	Macro Definition Documentation	1702
74.3.5	Enumeration Type Documentation	1702
74.3.6	Function Documentation	1702

Section No.	Title	Page No.
74.4	MMC Card Driver	1712
74.4.1	Overview	1712
74.4.2	MMC CARD Operation	1712
74.4.3	Data Structure Documentation	1714
74.4.4	Macro Definition Documentation	1716
74.4.5	Enumeration Type Documentation	1716
74.4.6	Function Documentation	1716
74.5	SDMMC HOST Driver	1730
74.5.1	Overview	1730
74.6	SDMMC OSA	1731
74.6.1	Overview	1731
74.6.2	Data Structure Documentation	1732
74.6.3	Function Documentation	1732
74.6.4	USDHC HOST adapter Driver	1736
74.7	SDMMC Common	1749
74.7.1	Overview	1749
74.7.2	Data Structure Documentation	1768
74.7.3	Macro Definition Documentation	1781
74.7.4	Enumeration Type Documentation	1781
74.7.5	Function Documentation	1798
Chapter 75 CODEC Driver		
75.1	Overview	1803
75.2	CODEC Common Driver	1804
75.2.1	Overview	1804
75.2.2	Data Structure Documentation	1809
75.2.3	Macro Definition Documentation	1810
75.2.4	Enumeration Type Documentation	1810
75.2.5	Function Documentation	1815
75.3	CODEC I2C Driver	1819
75.3.1	Overview	1819
75.3.2	Data Structure Documentation	1820
75.3.3	Enumeration Type Documentation	1820
75.3.4	Function Documentation	1820
75.4	WM8960 Driver	1823
75.4.1	Overview	1823
75.4.2	Data Structure Documentation	1826
75.4.3	Macro Definition Documentation	1828
75.4.4	Enumeration Type Documentation	1828

Section No.	Title	Page No.
75.4.5	Function Documentation	1830
75.4.6	WM8960 Adapter	1837
Chapter 76 Serial Manager		
76.1	Overview	1845
76.2	Data Structure Documentation	1848
76.2.1	struct serial_manager_config_t	1848
76.2.2	struct serial_manager_callback_message_t	1848
76.3	Macro Definition Documentation	1848
76.3.1	SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE	1849
76.3.2	SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE	1849
76.3.3	SERIAL_MANAGER_USE_COMMON_TASK	1849
76.3.4	SERIAL_MANAGER_HANDLE_SIZE	1849
76.3.5	SERIAL_MANAGER_HANDLE_DEFINE	1849
76.3.6	SERIAL_MANAGER_WRITE_HANDLE_DEFINE	1849
76.3.7	SERIAL_MANAGER_READ_HANDLE_DEFINE	1850
76.3.8	SERIAL_MANAGER_TASK_PRIORITY	1850
76.3.9	SERIAL_MANAGER_TASK_STACK_SIZE	1850
76.4	Enumeration Type Documentation	1850
76.4.1	serial_port_type_t	1850
76.4.2	serial_manager_type_t	1851
76.4.3	serial_manager_status_t	1851
76.5	Function Documentation	1851
76.5.1	SerialManager_Init	1851
76.5.2	SerialManager_Deinit	1852
76.5.3	SerialManager_OpenWriteHandle	1853
76.5.4	SerialManager_CloseWriteHandle	1854
76.5.5	SerialManager_OpenReadHandle	1854
76.5.6	SerialManager_CloseReadHandle	1855
76.5.7	SerialManager_WriteBlocking	1856
76.5.8	SerialManager_ReadBlocking	1856
76.5.9	SerialManager_EnterLowpower	1857
76.5.10	SerialManager_ExitLowpower	1857
76.5.11	SerialManager_needPollingIsr	1858
76.6	Serial Port Uart	1859
76.6.1	Overview	1859
76.6.2	Enumeration Type Documentation	1859
76.7	Serial Port USB	1860
76.7.1	Overview	1860

Section No.	Title	Page No.
76.7.2	Data Structure Documentation	1860
76.7.3	Enumeration Type Documentation	1861
76.7.4	USB Device Configuration	1862
76.8	Serial Port SWO	1863
76.8.1	Overview	1863
76.8.2	Data Structure Documentation	1863
76.8.3	Enumeration Type Documentation	1863
 Chapter 77 Gpc		
77.1	Overview	1864
77.2	Data Structure Documentation	1868
77.2.1	struct gpc_tran_step_config_t	1869
77.3	Macro Definition Documentation	1869
77.3.1	FSL_GPC_RIVER_VERSION	1869
77.4	Enumeration Type Documentation	1869
77.4.1	anonymous enum	1869
77.4.2	anonymous enum	1869
77.4.3	gpc_cm_standby_mode_status_t	1870
77.4.4	gpc_cm_tran_step_t	1870
77.4.5	gpc_tran_step_counter_mode_t	1870
77.4.6	gpc_sp_tran_step_t	1871
77.4.7	gpc_cpu_mode_t	1871
77.4.8	gpc_cm_wakeup_sp_sel_t	1872
77.4.9	gpc_stby_tran_step_t	1872
77.5	Function Documentation	1872
77.5.1	GPC_CM_SetNextCpuMode	1872
77.5.2	GPC_CM_GetCurrentCpuMode	1873
77.5.3	GPC_CM_GetPreviousCpuMode	1873
77.5.4	GPC_CM_EnableIrqWakeup	1873
77.5.5	GPC_CM_EnableNonIrqWakeup	1874
77.5.6	GPC_CM_GetIrqWakeupStatus	1874
77.5.7	GPC_CM_GetNonIrqWakeupStatus	1874
77.5.8	GPC_CM_ConfigCpuModeTransitionStep	1875
77.5.9	GPC_CM_RequestSleepModeSetPointTransition	1875
77.5.10	GPC_CM_RequestRunModeSetPointTransition	1876
77.5.11	GPC_CM_SetSetPointMapping	1876
77.5.12	GPC_CM_SetCpuModeSetPointMapping	1876
77.5.13	GPC_CM_RequestStandbyMode	1878
77.5.14	GPC_CM_ClearStandbyModeRequest	1878

Section No.	Title	Page No.
77.5.15	GPC_CM_GetStandbyModeStatus	1878
77.5.16	GPC_CM_GetInterruptStatusFlags	1879
77.5.17	GPC_CM_ClearInterruptStatusFlags	1880
77.5.18	GPC_SP_SetSetpointPriority	1880
77.5.19	GPC_SP_ConfigSetPointTransitionStep	1880
77.5.20	GPC_SP_GetCurrentSetPoint	1881
77.5.21	GPC_SP_GetPreviousSetPoint	1881
77.5.22	GPC_SP_GetTargetSetPoint	1881
77.5.23	GPC_STBY_ConfigStandbyTransitionStep	1882

Chapter 78 PGMC

78.1	Overview	1883
78.2	Data Structure Documentation	1887
78.2.1	struct pgmc_bpc_cpu_power_mode_option_t	1887
78.2.2	struct pgmc_bpc_setpoint_mode_option_t	1887
78.3	Macro Definition Documentation	1888
78.3.1	FSL_PGMC_RIVER_VERSION	1888
78.4	Enumeration Type Documentation	1888
78.4.1	anonymous enum	1888
78.4.2	_pgmc_mif_signal_behaviour	1888
78.4.3	pgmc_bpc_assign_domain_t	1889
78.4.4	pgmc_cpu_mode_t	1889
78.4.5	pgmc_control_mode_t	1889
78.4.6	pgmc_memory_low_power_level_t	1889
78.4.7	pgmc_mif_signal_t	1890
78.5	Function Documentation	1890
78.5.1	PGMC_BPC_ControlPowerDomainByCpuPowerMode	1890
78.5.2	PGMC_BPC_ControlPowerDomainBySetPointMode	1890
78.5.3	PGMC_BPC_ControlPowerDomainBySoftwareMode	1891
78.5.4	PGMC_BPC_DisableLowPower	1891
78.5.5	PGMC_BPC_RequestStateRestoreAtRunMode	1891
78.5.6	PGMC_BPC_RequestStateRestoreAtSetPoint	1892
78.5.7	PGMC_BPC_AllowUserModeAccess	1892
78.5.8	PGMC_BPC_AllowNonSecureModeAccess	1892
78.5.9	PGMC_BPC_LockAccessSetting	1893
78.5.10	PGMC_BPC_SetDomainIdWhiteList	1893
78.5.11	PGMC_BPC_LockDomainIDWhiteList	1893
78.5.12	PGMC_BPC_LockLowPowerConfigurationFields	1894
78.5.13	PGMC_CPC_CORE_PowerOffByCpuPowerMode	1894
78.5.14	PGMC_CPC_CORE_PowerOffBySoftwareMode	1894

Section No.	Title	Page No.
78.5.15	PGMC_CPC_CORE_DisableLowPower	1894
78.5.16	PGMC_CPC_CACHE_ControlByCpuPowerMode	1895
78.5.17	PGMC_CPC_CACHE_ControlBySetPointMode	1895
78.5.18	PGMC_CPC_CACHE_DisableLowPower	1895
78.5.19	PGMC_CPC_CACHE_TriggerMLPLSoftwareChange	1896
78.5.20	PGMC_CPC_LMEM_ControlByCpuPowerMode	1896
78.5.21	PGMC_CPC_LMEM_ControlBySetPointMode	1896
78.5.22	PGMC_CPC_LMEM_DisableLowPower	1897
78.5.23	PGMC_CPC_LMEM_TriggerMLPLSoftwareChange	1897
78.5.24	PGMC_CPC_AllowUserModeAccess	1897
78.5.25	PGMC_CPC_AllowNonSecureModeAccess	1898
78.5.26	PGMC_CPC_LockAccessSetting	1898
78.5.27	PGMC_CPC_SetDomainIdWhiteList	1898
78.5.28	PGMC_CPC_LockDomainIDWhiteList	1899
78.5.29	PGMC_CPC_LockLowPowerConfigurationFields	1899
78.5.30	PGMC_MIF_SetSignalBehaviour	1899
78.5.31	PGMC_MIF_LockLowPowerConfigurationFields	1900
78.5.32	PGMC_PPC_TriggerPMICStandbySoftMode	1900
78.5.33	PGMC_PPC_ControlByCpuPowerMode	1900
78.5.34	PGMC_PPC_ControlBySetPointMode	1900
78.5.35	PGMC_PPC_DisableLowPower	1901
78.5.36	PGMC_PPC_AllowUserModeAccess	1901
78.5.37	PGMC_PPC_AllowNonSecureModeAccess	1901
78.5.38	PGMC_PPC_LockAccessSetting	1902
78.5.39	PGMC_PPC_SetDomainIdWhiteList	1902
78.5.40	PGMC_PPC_LockDomainIDWhiteList	1902
78.5.41	PGMC_PPC_LockLowPowerConfigurationFields	1903

Chapter 79 Soc_mipi_csi2rx

79.1	Overview	1904
79.2	Macro Definition Documentation	1904
79.2.1	FSL_SOC_MIPI_CSI2RX_DRIVER_VERSION	1904
79.3	Function Documentation	1904
79.3.1	MIPI_CSI2RX_SoftwareReset	1904
79.3.2	MIPI_CSI2RX_InitInterface	1904
79.3.3	MIPI_CSI2RX_DeinitInterface	1905

Chapter 80 Asrc_edma_driver

80.1	Overview	1906
80.2	Data Structure Documentation	1907

Section No.	Title	Page No.
80.2.1	struct asrc_p2p_edma_config_t	1907
80.2.2	struct asrc_in_edma_handle_t	1907
80.2.3	struct asrc_out_edma_handle_t	1908
80.2.4	struct _asrc_edma_handle	1909
80.3	Macro Definition Documentation	1909
80.3.1	ASRC_XFER_IN_QUEUE_SIZE	1909
80.4	Function Documentation	1909
80.4.1	ASRC_TransferInCreateHandleEDMA	1909
80.4.2	ASRC_TransferOutCreateHandleEDMA	1910
80.4.3	ASRC_TransferSetChannelPairConfigEDMA	1910
80.4.4	ASRC_GetOutSamplesSizeEDMA	1911
80.4.5	ASRC_TransferEDMA	1911
80.4.6	ASRC_TransferInAbortEDMA	1912
80.4.7	ASRC_TransferOutAbortEDMA	1912
80.4.8	ASRC_TransferInTerminalEDMA	1912
80.4.9	ASRC_TransferOutTerminalEDMA	1913
 Chapter 81 Enet_qos_qos		
81.1	Overview	1914
81.2	Data Structure Documentation	1924
81.2.1	struct enet_qos_rx_bd_struct_t	1924
81.2.2	struct enet_qos_tx_bd_struct_t	1925
81.2.3	struct enet_qos_ptp_time_t	1925
81.2.4	struct enet_qos_frame_info_t	1925
81.2.5	struct enet_qos_tx_dirty_ring_t	1926
81.2.6	struct enet_qos_ptp_config_t	1926
81.2.7	struct enet_qos_est_gate_op_t	1927
81.2.8	struct enet_qos_est_gcl_t	1927
81.2.9	struct enet_qos_rxp_config_t	1927
81.2.10	struct enet_qos_buffer_config_t	1928
81.2.11	struct enet_qos_cbs_config_t	1929
81.2.12	struct enet_qos_queue_tx_config_t	1930
81.2.13	struct enet_qos_queue_rx_config_t	1930
81.2.14	struct enet_qos_multiqueue_config_t	1931
81.2.15	struct enet_qos_config_t	1931
81.2.16	struct enet_qos_tx_bd_ring_t	1932
81.2.17	struct enet_qos_rx_bd_ring_t	1933
81.2.18	struct _enet_qos_handle	1933
81.2.19	struct enet_qos_buffer_struct_t	1935
81.2.20	struct enet_qos_rx_frame_error_t	1935
81.2.21	struct enet_qos_rx_frame_struct_t	1936

Section No.	Title	Page No.
81.2.22	struct enet_qos_transfer_stats_t	1936
81.3	Macro Definition Documentation	1937
81.3.1	FSL_ENET_QOS_DRIVER_VERSION	1937
81.3.2	ENET_QOS_RXDESCRIP_RD_BUFF1VALID_MASK	1937
81.3.3	ENET_QOS_RXDESCRIP_RD_BUFF2VALID_MASK	1937
81.3.4	ENET_QOS_RXDESCRIP_RD_IOC_MASK	1937
81.3.5	ENET_QOS_RXDESCRIP_RD_OWN_MASK	1937
81.3.6	ENET_QOS_RXDESCRIP_WR_ERR_MASK	1937
81.3.7	ENET_QOS_TXDESCRIP_RD_BL1_MASK	1937
81.3.8	ENET_QOS_TXDESCRIP_WB_TTSS_MASK	1937
81.3.9	ENET_QOS_RING_NUM_MAX	1937
81.3.10	ENET_QOS_FRAME_MAX_FRAMELEN	1937
81.3.11	ENET_QOS_FCS_LEN	1937
81.3.12	ENET_QOS_ADDR_ALIGNMENT	1937
81.3.13	ENET_QOS_BUFF_ALIGNMENT	1938
81.3.14	ENET_QOS_MTL_RXFIFOSIZE	1938
81.3.15	ENET_QOS_MTL_TXFIFOSIZE	1938
81.3.16	ENET_QOS_MACINT_ENUM_OFFSET	1938
81.4	Typedef Documentation	1938
81.4.1	enet_qos_rx_alloc_callback_t	1938
81.4.2	enet_qos_rx_free_callback_t	1938
81.4.3	enet_qos_callback_t	1938
81.5	Enumeration Type Documentation	1938
81.5.1	anonymous enum	1938
81.5.2	enet_qos_mii_mode_t	1939
81.5.3	enet_qos_mii_speed_t	1939
81.5.4	enet_qos_mii_duplex_t	1939
81.5.5	enet_qos_mii_normal_opcode	1939
81.5.6	enet_qos_dma_burstlen	1939
81.5.7	enet_qos_desc_flag	1940
81.5.8	enet_qos_systime_op	1940
81.5.9	enet_qos_ts_rollover_type	1940
81.5.10	enet_qos_special_config_t	1940
81.5.11	enet_qos_dma_interrupt_enable_t	1941
81.5.12	enet_qos_mac_interrupt_enable_t	1941
81.5.13	enet_qos_event_t	1941
81.5.14	enet_qos_queue_mode_t	1942
81.5.15	enet_qos_mtl_multiqueue_txsche	1942
81.5.16	enet_qos_mtl_multiqueue_rxsche	1942
81.5.17	enet_qos_mtl_rxqueueuemap_t	1942
81.5.18	enet_qos_rx_queue_route_t	1942
81.5.19	enet_qos_ptp_event_type_t	1942

Section No.	Title	Page No.
81.5.20	enet_qos_ptp_pps_instance_t	1943
81.5.21	enet_qos_ptp_pps_trgt_mode_t	1943
81.5.22	enet_qos_ptp_pps_cmd_t	1943
81.5.23	enet_qos_ets_list_length_t	1943
81.5.24	enet_qos_ets_gccr_addr_t	1944
81.5.25	enet_qos_rxp_dma_chn_t	1944
81.6	Function Documentation	1944
81.6.1	ENET_QOS_SetSYSControl	1944
81.6.2	ENET_QOS_GetDefaultConfig	1944
81.6.3	ENET_QOS_Up	1945
81.6.4	ENET_QOS_Init	1945
81.6.5	ENET_QOS_Down	1946
81.6.6	ENET_QOS_Deinit	1946
81.6.7	ENET_QOS_GetInstance	1946
81.6.8	ENET_QOS_DescriptorInit	1946
81.6.9	ENET_QOS_RxBufferAllocAll	1947
81.6.10	ENET_QOS_RxBufferFreeAll	1947
81.6.11	ENET_QOS_StartRxTx	1948
81.6.12	ENET_QOS_SetMII	1949
81.6.13	ENET_QOS_SetSMI	1949
81.6.14	ENET_QOS_IsSMIBusy	1949
81.6.15	ENET_QOS_ReadSMIData	1950
81.6.16	ENET_QOS_StartSMIRead	1950
81.6.17	ENET_QOS_StartSMIWrite	1950
81.6.18	ENET_QOS_StartExtC45SMIWrite	1951
81.6.19	ENET_QOS_StartExtC45SMIRead	1951
81.6.20	ENET_QOS_SetMacAddr	1951
81.6.21	ENET_QOS_GetMacAddr	1952
81.6.22	ENET_QOS_AddMulticastGroup	1952
81.6.23	ENET_QOS_LeaveMulticastGroup	1952
81.6.24	ENET_QOS_AcceptAllMulticast	1952
81.6.25	ENET_QOS_RejectAllMulticast	1953
81.6.26	ENET_QOS_EnterPowerDown	1953
81.6.27	ENET_QOS_ExitPowerDown	1953
81.6.28	ENET_QOS_EnableRxParser	1953
81.6.29	ENET_QOS_EnableInterrupts	1954
81.6.30	ENET_QOS_DisableInterrupts	1954
81.6.31	ENET_QOS_GetDmaInterruptStatus	1955
81.6.32	ENET_QOS_ClearDmaInterruptStatus	1955
81.6.33	ENET_QOS_GetMacInterruptStatus	1955
81.6.34	ENET_QOS_ClearMacInterruptStatus	1956
81.6.35	ENET_QOS_IsTxDescriptorDmaOwn	1956
81.6.36	ENET_QOS_SetupTxDescriptor	1957
81.6.37	ENET_QOS_UpdateTxDescriptorTail	1958

Section No.	Title	Page No.
81.6.38	ENET_QOS_UpdateRxDescriptorTail	1958
81.6.39	ENET_QOS_GetRxDescriptor	1959
81.6.40	ENET_QOS_UpdateRxDescriptor	1959
81.6.41	ENET_QOS_ConfigureRxParser	1960
81.6.42	ENET_QOS_ReadRxParser	1960
81.6.43	ENET_QOS_EstProgramGcl	1961
81.6.44	ENET_QOS_EstReadGcl	1962
81.6.45	ENET_QOS_FpeEnable	1962
81.6.46	ENET_QOS_FpeDisable	1962
81.6.47	ENET_QOS_FpeConfigPreemptable	1963
81.6.48	ENET_QOS_AVBConfigure	1964
81.6.49	ENET_QOS_GetStatistics	1964
81.6.50	ENET_QOS_CreateHandler	1964
81.6.51	ENET_QOS_GetRxFrameSize	1965
81.6.52	ENET_QOS_ReadFrame	1966
81.6.53	ENET_QOS_SendFrame	1966
81.6.54	ENET_QOS_ReclaimTxDescriptor	1967
81.6.55	ENET_QOS_CommonIRQHandler	1967
81.6.56	ENET_QOS_SetISRHandler	1968
81.6.57	ENET_QOS_Ptp1588CorrectTimerInCoarse	1968
81.6.58	ENET_QOS_Ptp1588CorrectTimerInFine	1968
81.6.59	ENET_QOS_Ptp1588GetAddend	1969
81.6.60	ENET_QOS_Ptp1588GetTimerNoIRQDisable	1970
81.6.61	ENET_Ptp1588PpsControl	1970
81.6.62	ENET_QOS_Ptp1588PpsSetTrgtTime	1970
81.6.63	ENET_QOS_Ptp1588PpsSetWidth	1971
81.6.64	ENET_QOS_Ptp1588PpsSetInterval	1971
81.6.65	ENET_QOS_Ptp1588GetTimer	1971
81.6.66	ENET_QOS_GetTxFrame	1972
81.6.67	ENET_QOS_GetRxFrame	1972
81.7	Variable Documentation	1973
81.7.1	s_enetqosClock	1973
 Chapter 82 Iee		
82.1	Overview	1974
82.2	Data Structure Documentation	1975
82.2.1	struct iee_config_t	1975
82.3	Macro Definition Documentation	1975
82.3.1	FSL_IEE_DRIVER_VERSION	1975
82.4	Enumeration Type Documentation	1976

Section No.	Title	Page No.
82.4.1	iee_region_t	1976
82.4.2	iee_aes_bypass_t	1976
82.4.3	iee_aes_mode_t	1976
82.4.4	iee_aes_key_size_t	1976
82.4.5	iee_aes_key_num_t	1977
82.5	Function Documentation	1977
82.5.1	IEE_Init	1977
82.5.2	IEE_GetDefaultConfig	1977
82.5.3	IEE_SetRegionConfig	1977
82.5.4	IEE_SetRegionKey	1978
82.5.5	IEE_GetOffset	1978
82.5.6	IEE_LockRegionConfig	1978
 Chapter 83 Ieer		
83.1	Overview	1980
83.2	Macro Definition Documentation	1980
83.2.1	FSL_IEE_APC_DRIVER_VERSION	1981
83.3	Enumeration Type Documentation	1981
83.3.1	iee_apc_region_t	1981
83.3.2	iee_apc_domain_t	1981
83.4	Function Documentation	1981
83.4.1	IEE_APC_GlobalEnable	1981
83.4.2	IEE_APC_GlobalDisable	1982
83.4.3	IEE_APC_SetRegionConfig	1982
83.4.4	IEE_APC_LockRegionConfig	1982
83.4.5	IEE_APC_RegionEnable	1983
 Chapter 84 Key_manager		
84.1	Overview	1984
84.2	Data Structure Documentation	1984
84.2.1	struct domain_slot_config_t	1984
84.3	Macro Definition Documentation	1985
84.3.1	FSL_KEYMGR_DRIVER_VERSION	1985
84.4	Function Documentation	1985
84.4.1	KEYMGR_MasterKeyControll	1985
84.4.2	KEYMGR_OTFAD1KeyControll	1986
84.4.3	KEYMGR_OTFAD2KeyControll	1986

Section No.	Title	Page No.
84.4.4	KEYMGR_IEEKeyReload	1987
84.4.5	KEYMGR_PUFKeyLock	1987
84.4.6	KEYMGR_SlotControl	1987
84.4.7	KEYMGR_Init	1988
84.4.8	KEYMGR_GetDefaultConfig	1989

Chapter 1

Introduction

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

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

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

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

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

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

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

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

MCUXpresso SDK Folder Structure

Chapter 2

Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

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

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

© 2021 NXP B.V.

Chapter 3

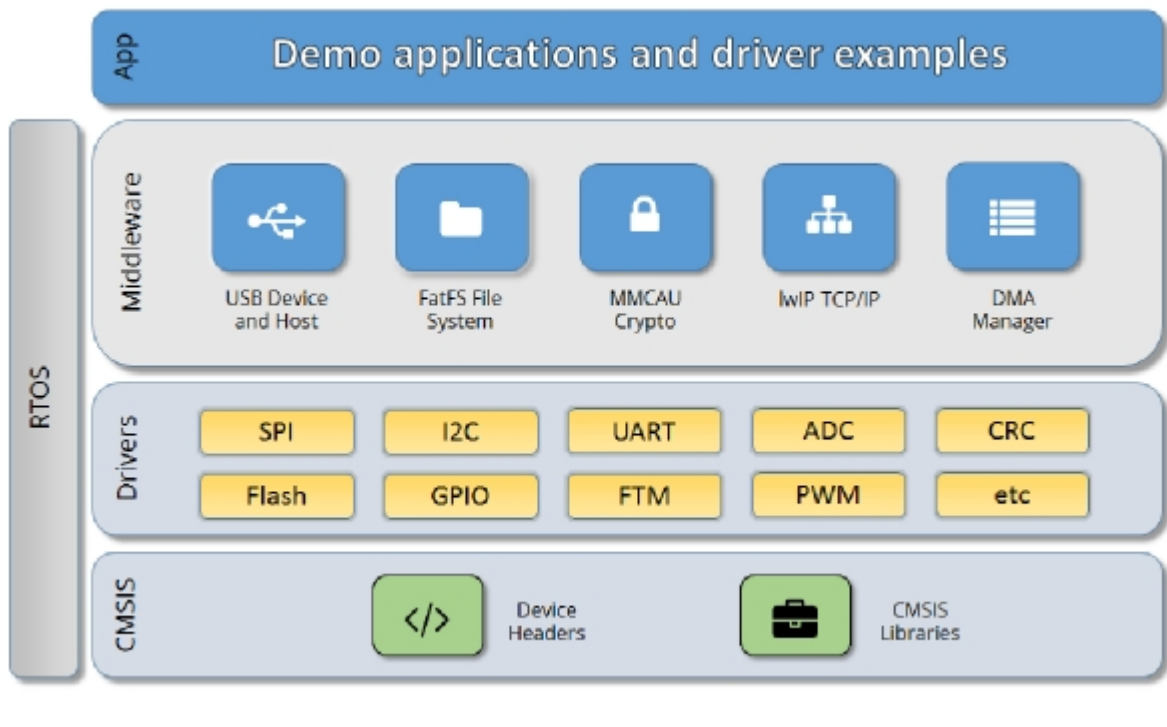
Architectural Overview

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

Overview

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

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



MCUXpresso SDK Block Diagram

MCU header files

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

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

CMSIS Support

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

MCUXpresso SDK Peripheral Drivers

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

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

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

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

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

Interrupt handling for transactional APIs

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

```
PUBWEAK SPI0_IRQHandler
PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

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

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

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

Feature Header Files

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

Application

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

Chapter 4

ANATOP_AI driver

4.1 Overview

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

Files

- file [fsl_anatop_ai.h](#)

Enumerations

- enum [anatop_ai_itf_t](#) {
 [kAI_Itf_Ldo](#) = 0,
 [kAI_Itf_1g](#) = 1,
 [kAI_Itf_Audio](#) = 2,
 [kAI_Itf_Video](#) = 3,
 [kAI_Itf_400m](#) = 4,
 [kAI_Itf_Temp](#) = 5,
 [kAI_Itf_Bandgap](#) = 6 }
 Anatop AI ITF enumeration.
- enum [anatop_ai_reg_t](#) {

```

kAI_PHY_LDO_CTRL0 = 0x0,
kAI_PHY_LDO_CTRL0_SET = 0x4,
kAI_PHY_LDO_CTRL0_CLR = 0x8,
kAI_PHY_LDO_CTRL0_TOG = 0xC,
kAI_PHY_LDO_STAT0 = 0x50,
kAI_PHY_LDO_STAT0_SET = 0x54,
kAI_PHY_LDO_STAT0_CLR = 0x58,
kAI_PHY_LDO_STAT0_TOG = 0x5C,
kAI_BANDGAP_CTRL0 = 0x0,
kAI_BANDGAP_STAT0 = 0x50,
kAI_RCOSC400M_CTRL0 = 0x0,
kAI_RCOSC400M_CTRL0_SET = 0x4,
kAI_RCOSC400M_CTRL0_CLR = 0x8,
kAI_RCOSC400M_CTRL0_TOG = 0xC,
kAI_RCOSC400M_CTRL1 = 0x10,
kAI_RCOSC400M_CTRL1_SET = 0x14,
kAI_RCOSC400M_CTRL1_CLR = 0x18,
kAI_RCOSC400M_CTRL1_TOG = 0x1C,
kAI_RCOSC400M_CTRL2 = 0x20,
kAI_RCOSC400M_CTRL2_SET = 0x24,
kAI_RCOSC400M_CTRL2_CLR = 0x28,
kAI_RCOSC400M_CTRL2_TOG = 0x2C,
kAI_RCOSC400M_CTRL3 = 0x30,
kAI_RCOSC400M_CTRL3_SET = 0x34,
kAI_RCOSC400M_CTRL3_CLR = 0x38,
kAI_RCOSC400M_CTRL3_TOG = 0x3C,
kAI_RCOSC400M_STAT0 = 0x50,
kAI_RCOSC400M_STAT0_SET = 0x54,
kAI_RCOSC400M_STAT0_CLR = 0x58,
kAI_RCOSC400M_STAT0_TOG = 0x5C,
kAI_RCOSC400M_STAT1 = 0x60,
kAI_RCOSC400M_STAT1_SET = 0x64,
kAI_RCOSC400M_STAT1_CLR = 0x68,
kAI_RCOSC400M_STAT1_TOG = 0x6C,
kAI_RCOSC400M_STAT2 = 0x70,
kAI_RCOSC400M_STAT2_SET = 0x74,
kAI_RCOSC400M_STAT2_CLR = 0x78,
kAI_RCOSC400M_STAT2_TOG = 0x7C,
kAI_PLL1G_CTRL0 = 0x0,
kAI_PLL1G_CTRL0_SET = 0x4,
kAI_PLL1G_CTRL0_CLR = 0x8,
kAI_PLL1G_CTRL1 = 0x10,
kAI_PLL1G_CTRL1_SET = 0x14,
kAI_PLL1G_CTRL1_CLR = 0x18,
kAI_PLL1G_CTRL2 = 0x20,
kAI_PLL1G_CTRL2_SET = 0x24,
kAI_PLL1G_CTRL2_CLR = 0x28,
kAI_PLL1G_CTRL3 = 0x30,
kAI_PLL1G_CTRL3_SET = 0x34,

```

```
kAI_PLLVIDEO_CTRL3_CLR = 0x38 }
```

The enumeration of ANATOP AI Register.

Driver version

- #define **FSL_ANATOP_AI_DRIVER_VERSION** (**MAKE_VERSION**(1, 0, 0))
Anatop AI driver version 1.0.0.

CTRL0 - CTRL0 Register

- #define **AI_PHY_LDO_CTRL0_LINREG_EN**(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_LINREG_EN_SHIFT) & AI_PHY_LDO_CTRL0_LINREG_EN_MASK)
- #define **AI_PHY_LDO_CTRL0_LINREG_EN_MASK** (0x1U)
- #define **AI_PHY_LDO_CTRL0_LINREG_EN_SHIFT** (0U)
- #define **AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS**(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS_SHIFT) & AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS_MASK)
- *LINREG_EN - LinReg master enable LinReg master enable.*
- #define **AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS_MASK** (0x2U)
- #define **AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS_SHIFT** (1U)
- #define **AI_PHY_LDO_CTRL0_LIMIT_EN**(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_LIMIT_EN_SHIFT) & AI_PHY_LDO_CTRL0_LIMIT_EN_MASK)
- *LINREG_PWRUPLOAD_DIS - LinReg power-up load disable 0b0..Internal pull-down enabled 0b1..Internal pull-down disabled.*
- #define **AI_PHY_LDO_CTRL0_LIMIT_EN_MASK** (0x4U)
- #define **AI_PHY_LDO_CTRL0_LIMIT_EN_SHIFT** (2U)
- #define **AI_PHY_LDO_CTRL0_OUTPUT_TRG**(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_OUTPUT_TRG_SHIFT) & AI_PHY_LDO_CTRL0_OUTPUT_TRG_MASK)
- *LINREG_LIMIT_EN - LinReg current limit enable LinReg current-limit enable.*
- #define **AI_PHY_LDO_CTRL0_OUTPUT_TRG_MASK** (0x1F0U)
- #define **AI_PHY_LDO_CTRL0_OUTPUT_TRG_SHIFT** (4U)
- #define **AI_PHY_LDO_CTRL0_PHY_ISO_B**(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_PHY_ISO_B_SHIFT) & AI_PHY_LDO_CTRL0_PHY_ISO_B_MASK)
- *LINREG_OUTPUT_TRG - LinReg output voltage target setting 0b00000..Set output voltage to x.xV 0b10000..Set output voltage to 1.0V 0b11111..Set output voltage to x.xV.*
- #define **AI_PHY_LDO_CTRL0_PHY_ISO_B_MASK** (0x8000U)
- #define **AI_PHY_LDO_CTRL0_PHY_ISO_B_SHIFT** (15U)
- #define **AI_BANDGAP_CTRL0_REFTOP_PWD**(x) (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_PWD_SHIFT) & AI_BANDGAP_CTRL0_REFTOP_PWD_MASK)
- #define **AI_BANDGAP_CTRL0_REFTOP_PWD_MASK** (0x1U)
- #define **AI_BANDGAP_CTRL0_REFTOP_PWD_SHIFT** (0U)
- #define **AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD**(x)
- *REFTOP_PWD - This bit fully powers down the bandgap module.*
- #define **AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD_MASK** (0x2U)
- #define **AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD_SHIFT** (1U)
- #define **AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP**(x) (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP_SHIFT) & AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP_MASK)
- *REFOP_LINREGREF_PWD - This bit powers down only the voltage reference output section of the bandgap.*
- #define **AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP_MASK** (0x4U)

- #define **AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP_SHIFT** (2U)
- #define **AI_BANDGAP_CTRL0_REFTOP_LOWPOWER(x)** (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_LOWPOWER_SHIFT)) & AI_BANDGAP_CTRL0_REFTOP_LOWPOWER_MASK)
- REFTOP_PWDVBGUP - This bit powers down the VBGUP detector of the bandgap without affecting any additional functionality.*
- #define **AI_BANDGAP_CTRL0_REFTOP_LOWPOWER_MASK** (0x8U)
- #define **AI_BANDGAP_CTRL0_REFTOP_LOWPOWER_SHIFT** (3U)
- #define **AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF(x)**
- REFTOP_LOWPOWER - This bit enables the low-power operation of the bandgap by cutting the bias currents in half to the main amplifiers.*
- #define **AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF_MASK** (0x10U)
- #define **AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF_SHIFT** (4U)
- #define **AI_BANDGAP_CTRL0_REFTOP_VBGADJ(x)** (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_VBGADJ_SHIFT)) & AI_BANDGAP_CTRL0_REFTOP_VBGADJ_MASK)
- REFTOP_SELFBIASOFF - Control bit to disable the self-bias circuit in the bandgap.*
- #define **AI_BANDGAP_CTRL0_REFTOP_VBGADJ_MASK** (0xE0U)
- #define **AI_BANDGAP_CTRL0_REFTOP_VBGADJ_SHIFT** (5U)
- #define **AI_BANDGAP_CTRL0_REFTOP_IBZTCADJ(x)** (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_IBZTCADJ_SHIFT)) & AI_BANDGAP_CTRL0_REFTOP_IBZTCADJ_MASK)
- REFTOP_VBGADJ - These bits allow the output VBG voltage of the bandgap to be trimmed 000 : nominal 001 : +10mV 010 : +20mV 011 : +30mV 100 : -10mV 101 : -20mV 110 : -30mV 111 : -40mV.*
- #define **AI_BANDGAP_CTRL0_REFTOP_IBZTCADJ_MASK** (0x1C00U)
- #define **AI_BANDGAP_CTRL0_REFTOP_IBZTCADJ_SHIFT** (10U)
- #define **AI_RCOSC400M_CTRL0_REF_CLK_DIV(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL0_REF_CLK_DIV_SHIFT)) & AI_RCOSC400M_CTRL0_REF_CLK_DIV_MASK)
- #define **AI_RCOSC400M_CTRL0_REF_CLK_DIV_MASK** (0x3F000000U)
- #define **AI_RCOSC400M_CTRL0_REF_CLK_DIV_SHIFT** (24U)
- #define **AI_PLL1G_CTRL0_HOLD_RING_OFF(x)** (((uint32_t)((uint32_t)(x)) << AI_PLL1G_CTRL0_HOLD_RING_OFF_SHIFT)) & AI_PLL1G_CTRL0_HOLD_RING_OFF_MASK)
- #define **AI_PLL1G_CTRL0_HOLD_RING_OFF_MASK** (0x2000UL)
- #define **AI_PLL1G_CTRL0_HOLD_RING_OFF_SHIFT** (13U)
- #define **AI_PLL1G_CTRL0_POWER_UP(x)** (((uint32_t)((uint32_t)(x)) << AI_PLL1G_CTRL0_POWER_UP_SHIFT)) & AI_PLL1G_CTRL0_POWER_UP_MASK)
- #define **AI_PLL1G_CTRL0_POWER_UP_MASK** (0x4000UL)
- #define **AI_PLL1G_CTRL0_POWER_UP_SHIFT** (14U)
- #define **AI_PLL1G_CTRL0_ENABLE(x)** (((uint32_t)((uint32_t)(x)) << AI_PLL1G_CTRL0_ENABLE_SHIFT)) & AI_PLL1G_CTRL0_ENABLE_MASK)
- #define **AI_PLL1G_CTRL0_ENABLE_MASK** (0x8000UL)
- #define **AI_PLL1G_CTRL0_ENABLE_SHIFT** (15U)
- #define **AI_PLL1G_CTRL0_BYPASS(x)** (((uint32_t)((uint32_t)(x)) << AI_PLL1G_CTRL0_BYPASS_SHIFT)) & AI_PLL1G_CTRL0_BYPASS_MASK)
- #define **AI_PLL1G_CTRL0_BYPASS_MASK** (0x10000UL)
- #define **AI_PLL1G_CTRL0_BYPASS_SHIFT** (16U)
- #define **AI_PLL1G_CTRL0_PLL_REG_EN(x)** (((uint32_t)((uint32_t)(x)) << AI_PLL1G_CTRL0_PLL_REG_EN_SHIFT)) & AI_PLL1G_CTRL0_PLL_REG_EN_MASK)
- #define **AI_PLL1G_CTRL0_PLL_REG_EN_MASK** (0x400000UL)
- #define **AI_PLL1G_CTRL0_PLL_REG_EN_SHIFT** (22U)

- `#define AI_PLLAUDIO_CTRL0_HOLD_RING_OFF(x) (((uint32_t)((uint32_t)(x)) << AI_PLLAUDIO_CTRL0_HOLD_RING_OFF_SHIFT)) & AI_PLLAUDIO_CTRL0_HOLD_RING_OFF_MASK)`
- `#define AI_PLLAUDIO_CTRL0_HOLD_RING_OFF_MASK (0x2000UL)`
- `#define AI_PLLAUDIO_CTRL0_HOLD_RING_OFF_SHIFT (13U)`
- `#define AI_PLLAUDIO_CTRL0_POWER_UP(x) (((uint32_t)((uint32_t)(x)) << AI_PLLAUDIO_CTRL0_POWER_UP_SHIFT)) & AI_PLLAUDIO_CTRL0_POWER_UP_MASK)`
- `#define AI_PLLAUDIO_CTRL0_POWER_UP_MASK (0x4000UL)`
- `#define AI_PLLAUDIO_CTRL0_POWER_UP_SHIFT (14U)`
- `#define AI_PLLAUDIO_CTRL0_ENABLE(x) (((uint32_t)((uint32_t)(x)) << AI_PLLAUDIO_CTRL0_ENABLE_SHIFT)) & AI_PLLAUDIO_CTRL0_ENABLE_MASK)`
- `#define AI_PLLAUDIO_CTRL0_ENABLE_MASK (0x8000UL)`
- `#define AI_PLLAUDIO_CTRL0_ENABLE_SHIFT (15U)`
- `#define AI_PLLAUDIO_CTRL0_BYPASS(x) (((uint32_t)((uint32_t)(x)) << AI_PLLAUDIO_CTRL0_BYPASS_SHIFT)) & AI_PLLAUDIO_CTRL0_BYPASS_MASK)`
- `#define AI_PLLAUDIO_CTRL0_BYPASS_MASK (0x10000UL)`
- `#define AI_PLLAUDIO_CTRL0_BYPASS_SHIFT (16U)`
- `#define AI_PLLAUDIO_CTRL0_PLL_REG_EN(x) (((uint32_t)((uint32_t)(x)) << AI_PLLAUDIO_CTRL0_PLL_REG_EN_SHIFT)) & AI_PLLAUDIO_CTRL0_PLL_REG_EN_MASK)`
- `#define AI_PLLAUDIO_CTRL0_PLL_REG_EN_MASK (0x400000UL)`
- `#define AI_PLLAUDIO_CTRL0_PLL_REG_EN_SHIFT (22U)`
- `#define AI_PLLVIDEO_CTRL0_HOLD_RING_OFF(x) (((uint32_t)((uint32_t)(x)) << AI_PLLVIDEO_CTRL0_HOLD_RING_OFF_SHIFT)) & AI_PLLVIDEO_CTRL0_HOLD_RING_OFF_MASK)`
- `#define AI_PLLVIDEO_CTRL0_HOLD_RING_OFF_MASK (0x2000UL)`
- `#define AI_PLLVIDEO_CTRL0_HOLD_RING_OFF_SHIFT (13U)`
- `#define AI_PLLVIDEO_CTRL0_POWER_UP(x) (((uint32_t)((uint32_t)(x)) << AI_PLLVIDEO_CTRL0_POWER_UP_SHIFT)) & AI_PLLVIDEO_CTRL0_POWER_UP_MASK)`
- `#define AI_PLLVIDEO_CTRL0_POWER_UP_MASK (0x4000UL)`
- `#define AI_PLLVIDEO_CTRL0_POWER_UP_SHIFT (14U)`
- `#define AI_PLLVIDEO_CTRL0_ENABLE(x) (((uint32_t)((uint32_t)(x)) << AI_PLLVIDEO_CTRL0_ENABLE_SHIFT)) & AI_PLLVIDEO_CTRL0_ENABLE_MASK)`
- `#define AI_PLLVIDEO_CTRL0_ENABLE_MASK (0x8000UL)`
- `#define AI_PLLVIDEO_CTRL0_ENABLE_SHIFT (15U)`
- `#define AI_PLLVIDEO_CTRL0_BYPASS(x) (((uint32_t)((uint32_t)(x)) << AI_PLLVIDEO_CTRL0_BYPASS_SHIFT)) & AI_PLLVIDEO_CTRL0_BYPASS_MASK)`
- `#define AI_PLLVIDEO_CTRL0_BYPASS_MASK (0x10000UL)`
- `#define AI_PLLVIDEO_CTRL0_BYPASS_SHIFT (16U)`
- `#define AI_PLLVIDEO_CTRL0_PLL_REG_EN(x) (((uint32_t)((uint32_t)(x)) << AI_PLLVIDEO_CTRL0_PLL_REG_EN_SHIFT)) & AI_PLLVIDEO_CTRL0_PLL_REG_EN_MASK)`
- `#define AI_PLLVIDEO_CTRL0_PLL_REG_EN_MASK (0x400000UL)`
- `#define AI_PLLVIDEO_CTRL0_PLL_REG_EN_SHIFT (22U)`

STAT0 - STAT0 Register

- `#define AI_PHY_LDO_STAT0_LINREG_STAT(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_STAT0_LINREG_STAT_SHIFT)) & AI_PHY_LDO_STAT0_LINREG_STAT_MASK)`
- `#define AI_PHY_LDO_STAT0_LINREG_STAT_MASK (0xFU)`
- `#define AI_PHY_LDO_STAT0_LINREG_STAT_SHIFT (0U)`
- `#define AI_BANDGAP_STAT0_REFTOP_VBGUP(x) (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_STAT0_REFTOP_VBGUP_SHIFT)) & AI_BANDGAP_STAT0_REFTOP_VBGUP_MASK)`

- #define **AI_BANDGAP_STAT0_REFTOP_VBGUP_MASK** (0x1U)
- #define **AI_BANDGAP_STAT0_REFTOP_VBGUP_SHIFT** (0U)
- #define **AI_RCOSC400M_STAT0_CLK1M_ERR(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_STAT0_CLK1M_ERR_SHIFT)) & AI_RCOSC400M_STAT0_CLK1M_ERR_MASK)
- #define **AI_RCOSC400M_STAT0_CLK1M_ERR_MASK** (0x1U)
- #define **AI_RCOSC400M_STAT0_CLK1M_ERR_SHIFT** (0U)

CTRL1 - CTRL1 Register

- #define **AI_RCOSC400M_CTRL1_HYST_MINUS(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL1_HYST_MINUS_SHIFT)) & AI_RCOSC400M_CTRL1_HYST_MINUS_MASK)
- #define **AI_RCOSC400M_CTRL1_HYST_MINUS_MASK** (0xFU)
- #define **AI_RCOSC400M_CTRL1_HYST_MINUS_SHIFT** (0U)
- #define **AI_RCOSC400M_CTRL1_HYST_PLUS(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL1_HYST_PLUS_SHIFT)) & AI_RCOSC400M_CTRL1_HYST_PLUS_MASK)
- #define **AI_RCOSC400M_CTRL1_HYST_PLUS_MASK** (0xF00U)
- #define **AI_RCOSC400M_CTRL1_HYST_PLUS_SHIFT** (8U)
- #define **AI_RCOSC400M_CTRL1_TARGET_COUNT(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL1_TARGET_COUNT_SHIFT)) & AI_RCOSC400M_CTRL1_TARGET_COUNT_MASK)
- #define **AI_RCOSC400M_CTRL1_TARGET_COUNT_MASK** (0xFFFF0000U)
- #define **AI_RCOSC400M_CTRL1_TARGET_COUNT_SHIFT** (16U)

CTRL2 - CTRL2 Register

- #define **AI_RCOSC400M_CTRL2_TUNE_BYP(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL2_TUNE_BYP_SHIFT)) & AI_RCOSC400M_CTRL2_TUNE_BYP_MASK)
- #define **AI_RCOSC400M_CTRL2_TUNE_BYP_MASK** (0x400U)
- #define **AI_RCOSC400M_CTRL2_TUNE_BYP_SHIFT** (10U)
- #define **AI_RCOSC400M_CTRL2_TUNE_EN(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL2_TUNE_EN_SHIFT)) & AI_RCOSC400M_CTRL2_TUNE_EN_MASK)
- #define **AI_RCOSC400M_CTRL2_TUNE_EN_MASK** (0x1000U)
- #define **AI_RCOSC400M_CTRL2_TUNE_EN_SHIFT** (12U)
- #define **AI_RCOSC400M_CTRL2_TUNE_START(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL2_TUNE_START_SHIFT)) & AI_RCOSC400M_CTRL2_TUNE_START_MASK)
- #define **AI_RCOSC400M_CTRL2_TUNE_START_MASK** (0x4000U)
- #define **AI_RCOSC400M_CTRL2_TUNE_START_SHIFT** (14U)
- #define **AI_RCOSC400M_CTRL2_OSC_TUNE_VAL(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL2_OSC_TUNE_VAL_SHIFT)) & AI_RCOSC400M_CTRL2_OSC_TUNE_VAL_MASK)
- #define **AI_RCOSC400M_CTRL2_OSC_TUNE_VAL_MASK** (0xFF000000U)
- #define **AI_RCOSC400M_CTRL2_OSC_TUNE_VAL_SHIFT** (24U)

CTRL3 - CTRL3 Register

- #define **AI_RCOSC400M_CTRL3_CLR_ERR(x)** (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL3_CLR_ERR_SHIFT)) & AI_RCOSC400M_CTRL3_CLR_ERR_MASK)
- #define **AI_RCOSC400M_CTRL3_CLR_ERR_MASK** (0x1U)
- #define **AI_RCOSC400M_CTRL3_CLR_ERR_SHIFT** (0U)

- `#define AI_RCOSC400M_CTRL3_EN_1M_CLK(x) (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL3_EN_1M_CLK_SHIFT)) & AI_RCOSC400M_CTRL3_EN_1M_CLK_MASK)`
- `#define AI_RCOSC400M_CTRL3_EN_1M_CLK_MASK (0x100U)`
- `#define AI_RCOSC400M_CTRL3_EN_1M_CLK_SHIFT (8U)`
- `#define AI_RCOSC400M_CTRL3_MUX_1M_CLK(x) (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL3_MUX_1M_CLK_SHIFT)) & AI_RCOSC400M_CTRL3_MUX_1M_CLK_MASK)`
- `#define AI_RCOSC400M_CTRL3_MUX_1M_CLK_MASK (0x400U)`
- `#define AI_RCOSC400M_CTRL3_MUX_1M_CLK_SHIFT (10U)`
- `#define AI_RCOSC400M_CTRL3_COUNT_1M_CLK(x) (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_CTRL3_COUNT_1M_CLK_SHIFT)) & AI_RCOSC400M_CTRL3_COUNT_1M_CLK_MASK)`
- `#define AI_RCOSC400M_CTRL3_COUNT_1M_CLK_MASK (0xFFFF0000U)`
- `#define AI_RCOSC400M_CTRL3_COUNT_1M_CLK_SHIFT (16U)`

STAT1 - STAT1 Register

- `#define AI_RCOSC400M_STAT1_CURR_COUNT_VAL(x) (((uint32_t)((uint32_t)(x)) << AI_RCOSC400M_STAT1_CURR_COUNT_VAL_SHIFT)) & AI_RCOSC400M_STAT1_CURR_COUNT_VAL_MASK)`
- `#define AI_RCOSC400M_STAT1_CURR_COUNT_VAL_MASK (0xFFFF0000U)`
- `#define AI_RCOSC400M_STAT1_CURR_COUNT_VAL_SHIFT (16U)`

STAT2 - STAT2 Register

- `#define AI_RCOSC400M_STAT2_CURR_OSC_TUNE_VAL(x)`
- `#define AI_RCOSC400M_STAT2_CURR_OSC_TUNE_VAL_MASK (0xFF000000U)`
- `#define AI_RCOSC400M_STAT2_CURR_OSC_TUNE_VAL_SHIFT (24U)`

4.2 Macro Definition Documentation

4.2.1 `#define FSL_ANATOP_AI_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`

4.2.2 `#define AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS_SHIFT)) & AI_PHY_LDO_CTRL0_PWRUPLOAD_DIS_MASK)`

Setting this bit will enable the regular

4.2.3 `#define AI_PHY_LDO_CTRL0_OUTPUT_TRG(x) (((uint32_t)((uint32_t)(x)) << AI_PHY_LDO_CTRL0_OUTPUT_TRG_SHIFT)) & AI_PHY_LDO_CTRL0_OUTPUT_TRG_MASK)`

Setting this bit will enable the current-limiter in the regulator

4.2.4 #define AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD(x)

Value:

```
((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD_SHIFT)) & \
AI_BANDGAP_CTRL0_REFTOP_LINREGREF_PWD_MASK)
```

Setting this bit high will disable reference output currents and voltages from the bandgap and will affect functionality and validity of the voltage detectors.

4.2.5 #define AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP(x) (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP_SHIFT)) & AI_BANDGAP_CTRL0_REFTOP_PWDVBGUP_MASK)

Setting this bit high will affect functionality and validity of the voltage detectors.

4.2.6 #define AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF(x)

Value:

```
((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF_SHIFT)) & \
AI_BANDGAP_CTRL0_REFTOP_SELFBIASOFF_MASK)
```

This will save power but could affect the accuracy of the output voltages and currents.

4.2.7 #define AI_BANDGAP_CTRL0_REFTOP_VBGADJ(x) (((uint32_t)((uint32_t)(x)) << AI_BANDGAP_CTRL0_REFTOP_VBGADJ_SHIFT)) & AI_BANDGAP_CTRL0_REFTOP_VBGADJ_MASK)

The self-bias circuit is used by the bandgap during startup. This bit should be set high after the bandgap has stabilized and is necessary for best noise performance of modules using the outputs of the bandgap. It is expected that this control bit be set low any time that either the bandgap is fully powered-down or the 1.8V supply is removed.

4.3 Enumeration Type Documentation

4.3.1 enum anatop_ai_itf_t

Enumerator

kAI_Itf_Ldo LDO ITF.
kAI_Itf_Ig 1G PLL ITF.

kAI_Itf_Audio Audio PLL ITF.
kAI_Itf_Video Video PLL ITF.
kAI_Itf_400m 400M OSC ITF.
kAI_Itf_Temp Temperature Sensor ITF.
kAI_Itf_Bandgap Bandgap ITF.

4.3.2 enum anatop_ai_reg_t

Enumerator

kAI_PHY_LDO_CTRL0 PHY LDO CTRL0 Register.
kAI_PHY_LDO_CTRL0_SET PHY LDO CTRL0 Set Register.
kAI_PHY_LDO_CTRL0_CLR PHY LDO CTRL0 Clr Register.
kAI_PHY_LDO_CTRL0_TOG PHY LDO CTRL0 TOG Register.
kAI_PHY_LDO_STAT0 PHY LDO STAT0 Register.
kAI_PHY_LDO_STAT0_SET PHY LDO STAT0 Set Register.
kAI_PHY_LDO_STAT0_CLR PHY LDO STAT0 Clr Register.
kAI_PHY_LDO_STAT0_TOG PHY LDO STAT0 Tog Register.
kAI_BANDGAP_CTRL0 BANDGAP CTRL0 Register.
kAI_BANDGAP_STAT0 BANDGAP STAT0 Register.
kAI_RCOSC400M_CTRL0 RC OSC 400M CTRL0 Register.
kAI_RCOSC400M_CTRL0_SET RC OSC 400M CTRL0 SET Register.
kAI_RCOSC400M_CTRL0_CLR RC OSC 400M CTRL0 CLR Register.
kAI_RCOSC400M_CTRL0_TOG RC OSC 400M CTRL0 TOG Register.
kAI_RCOSC400M_CTRL1 RC OSC 400M CTRL1 Register.
kAI_RCOSC400M_CTRL1_SET RC OSC 400M CTRL1 SET Register.
kAI_RCOSC400M_CTRL1_CLR RC OSC 400M CTRL1 CLR Register.
kAI_RCOSC400M_CTRL1_TOG RC OSC 400M CTRL1 TOG Register.
kAI_RCOSC400M_CTRL2 RC OSC 400M CTRL2 Register.
kAI_RCOSC400M_CTRL2_SET RC OSC 400M CTRL2 SET Register.
kAI_RCOSC400M_CTRL2_CLR RC OSC 400M CTRL2 CLR Register.
kAI_RCOSC400M_CTRL2_TOG RC OSC 400M CTRL2 TOG Register.
kAI_RCOSC400M_CTRL3 RC OSC 400M CTRL3 Register.
kAI_RCOSC400M_CTRL3_SET RC OSC 400M CTRL3 SET Register.
kAI_RCOSC400M_CTRL3_CLR RC OSC 400M CTRL3 CLR Register.
kAI_RCOSC400M_CTRL3_TOG RC OSC 400M CTRL3 TOG Register.
kAI_RCOSC400M_STAT0 RC OSC 400M STAT0 Register.
kAI_RCOSC400M_STAT0_SET RC OSC 400M STAT0 SET Register.
kAI_RCOSC400M_STAT0_CLR RC OSC 400M STAT0 CLR Register.
kAI_RCOSC400M_STAT0_TOG RC OSC 400M STAT0 TOG Register.
kAI_RCOSC400M_STAT1 RC OSC 400M STAT1 Register.
kAI_RCOSC400M_STAT1_SET RC OSC 400M STAT1 SET Register.
kAI_RCOSC400M_STAT1_CLR RC OSC 400M STAT1 CLR Register.
kAI_RCOSC400M_STAT1_TOG RC OSC 400M STAT1 TOG Register.

kAI_RCOSC400M_STAT2 RC OSC 400M STAT2 Register.
kAI_RCOSC400M_STAT2_SET RC OSC 400M STAT2 SET Register.
kAI_RCOSC400M_STAT2_CLR RC OSC 400M STAT2 CLR Register.
kAI_RCOSC400M_STAT2_TOG RC OSC 400M STAT2 TOG Register.
kAI_PLL1G_CTRL0 1G PLL CTRL0 Register.
kAI_PLL1G_CTRL0_SET 1G PLL CTRL0 SET Register.
kAI_PLL1G_CTRL0_CLR 1G PLL CTRL0 CLR Register.
kAI_PLL1G_CTRL1 1G PLL CTRL1 Register.
kAI_PLL1G_CTRL1_SET 1G PLL CTRL1 SET Register.
kAI_PLL1G_CTRL1_CLR 1G PLL CTRL1 CLR Register.
kAI_PLL1G_CTRL2 1G PLL CTRL2 Register.
kAI_PLL1G_CTRL2_SET 1G PLL CTRL2 SET Register.
kAI_PLL1G_CTRL2_CLR 1G PLL CTRL2 CLR Register.
kAI_PLL1G_CTRL3 1G PLL CTRL3 Register.
kAI_PLL1G_CTRL3_SET 1G PLL CTRL3 SET Register.
kAI_PLL1G_CTRL3_CLR 1G PLL CTRL3 CLR Register.
kAI_PLLAUDIO_CTRL0 AUDIO PLL CTRL0 Register.
kAI_PLLAUDIO_CTRL0_SET AUDIO PLL CTRL0 SET Register.
kAI_PLLAUDIO_CTRL0_CLR AUDIO PLL CTRL0 CLR Register.
kAI_PLLAUDIO_CTRL1 AUDIO PLL CTRL1 Register.
kAI_PLLAUDIO_CTRL1_SET AUDIO PLL CTRL1 SET Register.
kAI_PLLAUDIO_CTRL1_CLR AUDIO PLL CTRL1 CLR Register.
kAI_PLLAUDIO_CTRL2 AUDIO PLL CTRL2 Register.
kAI_PLLAUDIO_CTRL2_SET AUDIO PLL CTRL2 SET Register.
kAI_PLLAUDIO_CTRL2_CLR AUDIO PLL CTRL2 CLR Register.
kAI_PLLAUDIO_CTRL3 AUDIO PLL CTRL3 Register.
kAI_PLLAUDIO_CTRL3_SET AUDIO PLL CTRL3 SET Register.
kAI_PLLAUDIO_CTRL3_CLR AUDIO PLL CTRL3 CLR Register.
kAI_PLLVIDEO_CTRL0 VIDEO PLL CTRL0 Register.
kAI_PLLVIDEO_CTRL0_SET VIDEO PLL CTRL0 SET Register.
kAI_PLLVIDEO_CTRL0_CLR VIDEO PLL CTRL0 CLR Register.
kAI_PLLVIDEO_CTRL1 VIDEO PLL CTRL1 Register.
kAI_PLLVIDEO_CTRL1_SET VIDEO PLL CTRL1 SET Register.
kAI_PLLVIDEO_CTRL1_CLR VIDEO PLL CTRL1 CLR Register.
kAI_PLLVIDEO_CTRL2 VIDEO PLL CTRL2 Register.
kAI_PLLVIDEO_CTRL2_SET VIDEO PLL CTRL2 SET Register.
kAI_PLLVIDEO_CTRL2_CLR VIDEO PLL CTRL2 CLR Register.
kAI_PLLVIDEO_CTRL3 VIDEO PLL CTRL3 Register.
kAI_PLLVIDEO_CTRL3_SET VIDEO PLL CTRL3 SET Register.
kAI_PLLVIDEO_CTRL3_CLR VIDEO PLL CTRL3 CLR Register.

Chapter 5

Clock Driver

5.1 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_group_config_t](#)
The structure used to configure clock group. [More...](#)
- 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 specturm configure Pll. [More...](#)
- struct [clock_sys_pll2_config_t](#)
PLL configure for Sys Pll2. [More...](#)
- struct [clock_sys_pll1_config_t](#)
PLL configure for Sys Pll1. [More...](#)
- struct [clock_video_pll_config_t](#)
PLL configuration for AUDIO and VIDEO. [More...](#)
- struct [clock_sys_pll1_gpc_config_t](#)
PLL configuration fro AUDIO PLL, SYSTEM PLL1 and VIDEO PLL. [More...](#)
- struct [clock_enet_pll_config_t](#)
PLL configuration for ENET. [More...](#)
- struct [clock_root_config_t](#)
Clock root configuration. [More...](#)
- struct [clock_root_setpoint_config_t](#)
Clock root configuration in SetPoint Mode. [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 [SYS_PLL1_FREQ](#) (1000000000UL)
- *SYS_PLL_FREQ frequency in Hz.*
- #define [LPADC_CLOCKS](#)
- *Clock gate name array for ADC.*
- #define [ADC_ETC_CLOCKS](#)
- *Clock gate name array for ADC.*
- #define [AOI_CLOCKS](#)
- *Clock gate name array for AOI.*
- #define [DCDC_CLOCKS](#)
- *Clock gate name array for DCDC.*
- #define [DCDC_CLOCKS](#)
- *Clock gate name array for DCDC.*
- #define [SRC_CLOCKS](#)
- *Clock gate name array for SRC.*
- #define [GPC_CLOCKS](#)
- *Clock gate name array for GPC.*
- #define [SSARC_CLOCKS](#)
- *Clock gate name array for SSARC.*
- #define [WDOG_CLOCKS](#)
- *Clock gate name array for WDOG.*
- #define [EWM_CLOCKS](#)
- *Clock gate name array for EWM.*
- #define [SEMA_CLOCKS](#)
- *Clock gate name array for Sema.*
- #define [MU_CLOCKS](#)
- *Clock gate name array for MU.*
- #define [EDMA_CLOCKS](#)
- *Clock gate name array for EDMA.*
- #define [FLEXRAM_CLOCKS](#)
- *Clock gate name array for FLEXRAM.*
- #define [LMEM_CLOCKS](#)
- *Clock gate name array for LMEM.*
- #define [FLEXSPI_CLOCKS](#)
- *Clock gate name array for FLEXSPI.*
- #define [RDC_CLOCKS](#)
- *Clock gate name array for RDC.*
- #define [SEMC_CLOCKS](#)
- *Clock ip name array for SEMC.*
- #define [XECC_CLOCKS](#)
- *Clock ip name array for XECC.*
- #define [IEE_CLOCKS](#)
- *Clock ip name array for IEE.*
- #define [KEYMANAGER_CLOCKS](#)
- *Clock ip name array for KEY_MANAGER.*
- #define [PUF_CLOCKS](#)
- *Clock ip name array for PUF.*

- #define [OCOTP_CLOCKS](#)
Clock ip name array for OCOTP.
- #define [CAAM_CLOCKS](#)
Clock ip name array for CAAM.
- #define [XBAR_CLOCKS](#)
Clock ip name array for XBAR.
- #define [IOMUXC_CLOCKS](#)
Clock ip name array for IOMUXC.
- #define [GPIO_CLOCKS](#)
Clock ip name array for GPIO.
- #define [KPP_CLOCKS](#)
Clock ip name array for KPP.
- #define [FLEXIO_CLOCKS](#)
Clock ip name array for FLEXIO.
- #define [DAC_CLOCKS](#)
Clock ip name array for DAC.
- #define [CMP_CLOCKS](#)
Clock ip name array for CMP.
- #define [PIT_CLOCKS](#)
Clock ip name array for PIT.
- #define [GPT_CLOCKS](#)
Clock ip name array for GPT.
- #define [TMR_CLOCKS](#)
Clock ip name array for QTIMER.
- #define [ENC_CLOCKS](#)
Clock ip name array for ENC.
- #define [PWM_CLOCKS](#)
Clock ip name array for PWM.
- #define [FLEXCAN_CLOCKS](#)
Clock ip name array for FLEXCAN.
- #define [LPUART_CLOCKS](#)
Clock ip name array for LPUART.
- #define [LPI2C_CLOCKS](#)
Clock ip name array for LPI2C.
- #define [LPSPI_CLOCKS](#)
Clock ip name array for LPSPI.
- #define [EMVSIM_CLOCKS](#)
Clock ip name array for EMVSIM.
- #define [ENET_CLOCKS](#)
Clock ip name array for ENET.
- #define [ENETQOS_CLOCKS](#)
Clock ip name array for ENET_QOS.
- #define [USB_CLOCKS](#)
Clock ip name array for USB.
- #define [CDOG_CLOCKS](#)
Clock ip name array for CDOG.
- #define [USDHC_CLOCKS](#)
Clock ip name array for USDHC.
- #define [ASRC_CLOCKS](#)
Clock ip name array for ASRC.
- #define [MQS_CLOCKS](#)

- *Clock ip name array for MQS.*
• #define [PDM_CLOCKS](#)
- *Clock ip name array for PDM.*
• #define [SPDIF_CLOCKS](#)
- *Clock ip name array for SPDIF.*
• #define [SAI_CLOCKS](#)
- *Clock ip name array for SAI.*
• #define [PXP_CLOCKS](#)
- *Clock ip name array for PXP.*
• #define [GPU2D_CLOCKS](#)
- *Clock ip name array for GPU2d.*
• #define [LCDIF_CLOCKS](#)
- *Clock ip name array for LCDIF.*
• #define [LCDIFV2_CLOCKS](#)
- *Clock ip name array for LCDIFV2.*
• #define [MIPI_DSI_HOST_CLOCKS](#)
- *Clock ip name array for MIPI_DSI.*
• #define [MIPI_CSI2RX_CLOCKS](#)
- *Clock ip name array for MIPI_CSI.*
• #define [CSI_CLOCKS](#)
- *Clock ip name array for CSI.*
• #define [DCIC_CLOCKS](#)
- *Clock ip name array for DCIC.*
• #define [DMAMUX_CLOCKS](#)
- *Clock ip name array for DMAMUX_CLOCKS.*
• #define [XBARA_CLOCKS](#)
- *Clock ip name array for XBARA.*
• #define [XBARB_CLOCKS](#)
- *Clock ip name array for XBARB.*
• #define [CLOCK_GetCoreSysClkFreq](#) [CLOCK_GetCpuClkFreq](#)
- *For compatible with other platforms without CCM.*

Enumerations

- enum `clock_lpcg_t` {
 - `kCLOCK_M7` = 0,
 - `kCLOCK_M4` = 1,
 - `kCLOCK_Sim_M7` = 2,
 - `kCLOCK_Sim_M` = 3,
 - `kCLOCK_Sim_Disb` = 4,
 - `kCLOCK_Sim_Per` = 5,
 - `kCLOCK_Sim_Lpsr` = 6,
 - `kCLOCK_Anadig` = 7,
 - `kCLOCK_Dcdc` = 8,
 - `kCLOCK_Src` = 9,
 - `kCLOCK_Ccm` = 10,
 - `kCLOCK_Gpc` = 11,
 - `kCLOCK_Ssarc` = 12,
 - `kCLOCK_Sim_R` = 13,
 - `kCLOCK_Wdog1` = 14,
 - `kCLOCK_Wdog2` = 15,
 - `kCLOCK_Wdog3` = 16,
 - `kCLOCK_Wdog4` = 17,
 - `kCLOCK_Ewm0` = 18,
 - `kCLOCK_Sema` = 19,
 - `kCLOCK_Mu_A` = 20,
 - `kCLOCK_Mu_B` = 21,
 - `kCLOCK_Edma` = 22,
 - `kCLOCK_Edma_Lpsr` = 23,
 - `kCLOCK_Romcp` = 24,
 - `kCLOCK_Ocram` = 25,
 - `kCLOCK_Flexram` = 26,
 - `kCLOCK_Lmem` = 27,
 - `kCLOCK_Flexspi1` = 28,
 - `kCLOCK_Flexspi2` = 29,
 - `kCLOCK_Rdc` = 30,
 - `kCLOCK_M7_Xrdc` = 31,
 - `kCLOCK_M4_Xrdc` = 32,
 - `kCLOCK_Semc` = 33,
 - `kCLOCK_Xecc` = 34,
 - `kCLOCK_Iee` = 35,
 - `kCLOCK_Key_Manager` = 36,
 - `kCLOCK_Puf` = 36,
 - `kCLOCK_Ocotp` = 37,
 - `kCLOCK_Snvs_Hp` = 38,
 - `kCLOCK_Snvs` = 39,
 - `kCLOCK_Caam` = 40,
 - `kCLOCK_Jtag_Mux` = 41,
 - `kCLOCK_Cstrace` = 42,
 - `kCLOCK_Xbar1` = 43,
 - `kCLOCK_Xbar2` = 44,
 - `kCLOCK_Xbar3` = 45,

`kCLOCK_IpInvalid }`

Clock LPCG index.

- `enum clock_name_t {`
 - `kCLOCK_OscRc16M = 0,`
 - `kCLOCK_OscRc48M = 1,`
 - `kCLOCK_OscRc48MDiv2 = 2,`
 - `kCLOCK_OscRc400M = 3,`
 - `kCLOCK_Osc24M = 4,`
 - `kCLOCK_Osc24MOut = 5,`
 - `kCLOCK_ArmPll = 6,`
 - `kCLOCK_ArmPllOut = 7,`
 - `kCLOCK_SysPll2 = 8,`
 - `kCLOCK_SysPll2Out = 9,`
 - `kCLOCK_SysPll2Pfd0 = 10,`
 - `kCLOCK_SysPll2Pfd1 = 11,`
 - `kCLOCK_SysPll2Pfd2 = 12,`
 - `kCLOCK_SysPll2Pfd3 = 13,`
 - `kCLOCK_SysPll3 = 14,`
 - `kCLOCK_SysPll3Out = 15,`
 - `kCLOCK_SysPll3Div2 = 16,`
 - `kCLOCK_SysPll3Pfd0 = 17,`
 - `kCLOCK_SysPll3Pfd1 = 18,`
 - `kCLOCK_SysPll3Pfd2 = 19,`
 - `kCLOCK_SysPll3Pfd3 = 20,`
 - `kCLOCK_SysPll1 = 21,`
 - `kCLOCK_SysPll1Out = 22,`
 - `kCLOCK_SysPll1Div2 = 23,`
 - `kCLOCK_SysPll1Div5 = 24,`
 - `kCLOCK_AudioPll = 25,`
 - `kCLOCK_AudioPllOut = 26,`
 - `kCLOCK_VideoPll = 27,`
 - `kCLOCK_VideoPllOut = 28,`
 - `kCLOCK_CpuClk,`
 - `kCLOCK_CoreSysClk }`

Clock name.

- `enum clock_root_t {`

```

kCLOCK_Root_M7 = 0,
kCLOCK_Root_M4 = 1,
kCLOCK_Root_Bus = 2,
kCLOCK_Root_Bus_Lpsr = 3,
kCLOCK_Root_Semc = 4,
kCLOCK_Root_Cssys = 5,
kCLOCK_Root_Cstrace = 6,
kCLOCK_Root_M4_Systick = 7,
kCLOCK_Root_M7_Systick = 8,
kCLOCK_Root_Adc1 = 9,
kCLOCK_Root_Adc2 = 10,
kCLOCK_Root_Acmp = 11,
kCLOCK_Root_Flexio1 = 12,
kCLOCK_Root_Flexio2 = 13,
kCLOCK_Root_Gpt1 = 14,
kCLOCK_Root_Gpt2 = 15,
kCLOCK_Root_Gpt3 = 16,
kCLOCK_Root_Gpt4 = 17,
kCLOCK_Root_Gpt5 = 18,
kCLOCK_Root_Gpt6 = 19,
kCLOCK_Root_Flexspi1 = 20,
kCLOCK_Root_Flexspi2 = 21,
kCLOCK_Root_Can1 = 22,
kCLOCK_Root_Can2 = 23,
kCLOCK_Root_Can3 = 24,
kCLOCK_Root_Lpuart1 = 25,
kCLOCK_Root_Lpuart2 = 26,
kCLOCK_Root_Lpuart3 = 27,
kCLOCK_Root_Lpuart4 = 28,
kCLOCK_Root_Lpuart5 = 29,
kCLOCK_Root_Lpuart6 = 30,
kCLOCK_Root_Lpuart7 = 31,
kCLOCK_Root_Lpuart8 = 32,
kCLOCK_Root_Lpuart9 = 33,
kCLOCK_Root_Lpuart10 = 34,
kCLOCK_Root_Lpuart11 = 35,
kCLOCK_Root_Lpuart12 = 36,
kCLOCK_Root_Lpi2c1 = 37,
kCLOCK_Root_Lpi2c2 = 38,
kCLOCK_Root_Lpi2c3 = 39,
kCLOCK_Root_Lpi2c4 = 40,
kCLOCK_Root_Lpi2c5 = 41,
kCLOCK_Root_Lpi2c6 = 42,
kCLOCK_Root_Lpspi1 = 43,
kCLOCK_Root_Lpspi2 = 44,
kCLOCK_Root_Lpspi3 = 45,
kCLOCK_Root_Lpspi4 = 46,
kCLOCK_Root_Lpspi5 = 47,
kCLOCK_Root_Lpspi6 = 48,

```

```
kCLOCK_Root_Cko2 = 78 }
```

Root clock index.

- enum `clock_root_mux_source_t` {

```

kCLOCK_M7_ClockRoot_MuxOscRc48MDiv2 = 0U,
kCLOCK_M7_ClockRoot_MuxOsc24MOut = 1U,
kCLOCK_M7_ClockRoot_MuxOscRc400M = 2U,
kCLOCK_M7_ClockRoot_MuxOscRc16M = 3U,
kCLOCK_M7_ClockRoot_MuxArmPllOut = 4U,
kCLOCK_M7_ClockRoot_MuxSysPll1Out = 5U,
kCLOCK_M7_ClockRoot_MuxSysPll3Out = 6U,
kCLOCK_M7_ClockRoot_MuxVideoPllOut = 7U,
kCLOCK_M4_ClockRoot_MuxOscRc48MDiv2 = 0U,
kCLOCK_M4_ClockRoot_MuxOsc24MOut = 1U,
kCLOCK_M4_ClockRoot_MuxOscRc400M = 2U,
kCLOCK_M4_ClockRoot_MuxOscRc16M = 3U,
kCLOCK_M4_ClockRoot_MuxSysPll3Pfd3 = 4U,
kCLOCK_M4_ClockRoot_MuxSysPll3Out = 5U,
kCLOCK_M4_ClockRoot_MuxSysPll2Out = 6U,
kCLOCK_M4_ClockRoot_MuxSysPll1Div5 = 7U,
kCLOCK_BUS_ClockRoot_MuxOscRc48MDiv2 = 0U,
kCLOCK_BUS_ClockRoot_MuxOsc24MOut = 1U,
kCLOCK_BUS_ClockRoot_MuxOscRc400M = 2U,
kCLOCK_BUS_ClockRoot_MuxOscRc16M = 3U,
kCLOCK_BUS_ClockRoot_MuxSysPll3Out = 4U,
kCLOCK_BUS_ClockRoot_MuxSysPll1Div5 = 5U,
kCLOCK_BUS_ClockRoot_MuxSysPll2Out = 6U,
kCLOCK_BUS_ClockRoot_MuxSysPll2Pfd3 = 7U,
kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc48MDiv2 = 0U,
kCLOCK_BUS_LPSR_ClockRoot_MuxOsc24MOut = 1U,
kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc400M = 2U,
kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc16M = 3U,
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll3Pfd3 = 4U,
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll3Out = 5U,
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll2Out = 6U,
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll1Div5 = 7U,
kCLOCK_SEMC_ClockRoot_MuxOscRc48MDiv2 = 0U,
kCLOCK_SEMC_ClockRoot_MuxOsc24MOut = 1U,
kCLOCK_SEMC_ClockRoot_MuxOscRc400M = 2U,
kCLOCK_SEMC_ClockRoot_MuxOscRc16M = 3U,
kCLOCK_SEMC_ClockRoot_MuxSysPll1Div5 = 4U,
kCLOCK_SEMC_ClockRoot_MuxSysPll2Out = 5U,
kCLOCK_SEMC_ClockRoot_MuxSysPll2Pfd1 = 6U,
kCLOCK_SEMC_ClockRoot_MuxSysPll3Pfd0 = 7U,
kCLOCK_CSSYS_ClockRoot_MuxOscRc48MDiv2 = 0U,
kCLOCK_CSSYS_ClockRoot_MuxOsc24MOut = 1U,
kCLOCK_CSSYS_ClockRoot_MuxOscRc400M = 2U,
kCLOCK_CSSYS_ClockRoot_MuxOscRc16M = 3U,
kCLOCK_CSSYS_ClockRoot_MuxSysPll3Div2 = 4U,
kCLOCK_CSSYS_ClockRoot_MuxSysPll1Div5 = 5U,
kCLOCK_CSSYS_ClockRoot_MuxSysPll2Pfd3 = 6U,
kCLOCK_CSSYS_ClockRoot_MuxSysPll2Pfd3 = 7U,
kCLOCK_CSTRACE_ClockRoot_MuxOscRc48MDiv2 = 0U,

```

`kCLOCK_CK02_ClockRoot_MuxAudioPllOut = 7U }`

The enumerator of clock roots' clock source mux value.

- enum `clock_group_t` {
`kCLOCK_Group_FlexRAM = 0,`
`kCLOCK_Group_MipiDsi = 1,`
`kCLOCK_Group_Last }`
Clock group enumeration.
- enum `clock_osc_t` {
`kCLOCK_RcOsc = 0U,`
`kCLOCK_XtalOsc = 1U }`
OSC 24M source 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_PllClkSrc24M = 0U,`
`kCLOCK_PllSrcClkPN = 1U }`
PLL clock source, bypass clock source also.
- enum `clock_pll_post_div_t` {
`kCLOCK_PllPostDiv2 = 0U,`
`kCLOCK_PllPostDiv4 = 1U,`
`kCLOCK_PllPostDiv8 = 2U,`
`kCLOCK_PllPostDiv1 = 3U }`
PLL post divider enumeration.
- enum `clock_pll_t` {
`kCLOCK_PllArm,`
`kCLOCK_PllSys1,`
`kCLOCK_PllSys2,`
`kCLOCK_PllSys3,`
`kCLOCK_PllAudio,`
`kCLOCK_PllVideo,`
`kCLOCK_PllInvalid = -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_16MOsc_source_t` {
`kCLOCK_16MOscSourceFrom16MOsc` = 0U,
`kCLOCK_16MOscSourceFrom24MOsc` = 1U }

The enumeration of 16MHz RC oscillator clock source.

- 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, `uint8_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.
- void **CLOCK_SetGroupConfig** (clock_group_t group, const clock_group_config_t *config)
Set the clock group configuration.
- 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.
- static uint32_t **CLOCK_GetM7Freq** (void)
Get the CCM CPU/core/system frequency.
- static uint32_t **CLOCK_GetM4Freq** (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_GPC_SetSysPll1OutputFreq** (const clock_sys_pll1_gpc_config_t *config)
Set System PLL1 output frequency in GPC mode.
- 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, 5, 0))
CLOCK driver version.
- #define **SDK_DEVICE_MAXIMUM_CPU_CLOCK_FREQUENCY** (400000000UL)

OSC operations

- static uint32_t **CLOCK_GetRtcFreq** (void)
Gets the RTC clock frequency.
- static void **CLOCK_OSC_SetOsc48MControlMode** (clock_control_mode_t controlMode)
Set the control mode of 48MHz RC oscillator.
- static void **CLOCK_OSC_EnableOsc48M** (bool enable)
Enable/disable 48MHz RC oscillator.
- static void **CLOCK_OSC_SetOsc48MDiv2ControlMode** (clock_control_mode_t controlMode)
Set the control mode of the 24MHz clock sourced from 48MHz RC oscillator.
- static void **CLOCK_OSC_EnableOsc48MDiv2** (bool enable)
Enable/disable the 24MHz clock sourced from 48MHz RC oscillator.
- static void **CLOCK_OSC_SetOsc24MControlMode** (clock_control_mode_t controlMode)
Set the control mode of 24MHz crystal 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.
- static void **CLOCK_OSC_SetOscRc400MControlMode** (clock_control_mode_t controlMode)
Set the control mode of 400MHz RC oscillator.
- 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.
- static void **CLOCK_OSC_SetOsc16MControlMode** (clock_control_mode_t controlMode)
Set the control mode of 16MHz crystal oscillator.
- void **CLOCK_OSC_SetOsc16MConfig** (clock_16MOsc_source_t source, bool enablePowerSave, bool enableClockOut)
Configure the 16MHz oscillator.

PLL/PFD operations

- void **CLOCK_SetPllBypass** (clock_pll_t pll, bool bypass)
PLL bypass setting.
- status_t **CLOCK_CalcAvPllFreq** (clock_av_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.
- void **CLOCK_GPC_SetAudioPllOutputFreq** (const clock_audio_pll_gpc_config_t *config)
Set Audio PLL output frequency in GPC mode.
- status_t **CLOCK_InitVideoPllWithFreq** (uint32_t freqInMhz, bool ssEnable, uint32_t ssRange, uint32_t ssMod)
Initializes the Video PLL with Specific Frequency (in Mhz).
- void **CLOCK_InitVideoPll** (const clock_video_pll_config_t *config)
Initialize the video PLL.
- void **CLOCK_DeinitVideoPll** (void)
De-initialize the Video PLL.
- void **CLOCK_GPC_SetVideoPllOutputFreq** (const clock_video_pll_gpc_config_t *config)
Set Video PLL output frequency in GPC mode.
- 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** (uint32_t obsSigIndex, uint32_t obsIndex)
- 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_LockControlMode` (`clock_name_t` name)
Lock low power and access control mode for this 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.
- static bool `CLOCK_OSCPLL_IsSetPointImplemented` (`clock_name_t` name)
Check whether this clock implement SetPoint control scheme.
- static void `CLOCK_OSCPLL_ControlByUnassignedMode` (`clock_name_t` name)
Set this clock works in Unassigned Mode.
- void `CLOCK_OSCPLL_ControlBySetPointMode` (`clock_name_t` name, `uint16_t` spValue, `uint16_t` stbyValue)
Set this clock works in SetPoint control Mode.
- void `CLOCK_OSCPLL_ControlByCpuLowPowerMode` (`clock_name_t` name, `uint8_t` domainId, `clock_level_t` level0, `clock_level_t` level1)
Set this clock works in CPU Low Power Mode.
- static void `CLOCK_OSCPLL_SetCurrentClockLevel` (`clock_name_t` name, `clock_level_t` level)
Set clock depend level for current accessing domain.
- static void `CLOCK_OSCPLL_ControlByDomainMode` (`clock_name_t` name, `uint8_t` domainId)
Set this clock works in Domain Mode.
- static void `CLOCK_ROOT_LockControlMode` (`clock_root_t` name)
Lock low power and access control mode for this clock.
- 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 bool `CLOCK_ROOT_IsSetPointImplemented` (`clock_root_t` name)
Check whether this clock implement SetPoint control scheme.
- static void `CLOCK_ROOT_ControlByUnassignedMode` (`clock_root_t` name)
Set this clock works in Unassigned Mode.
- static void `CLOCK_ROOT_ConfigSetPoint` (`clock_root_t` name, `uint16_t` spIndex, const `clock_root_setpoint_config_t` *config)
Configure one SetPoint for this clock.
- static void `CLOCK_ROOT_EnableSetPointControl` (`clock_root_t` name)
Enable SetPoint control for this clock root.
- void `CLOCK_ROOT_ControlBySetPointMode` (`clock_root_t` name, const `clock_root_setpoint_config_t` *spTable)
Set this clock works in SetPoint controlled Mode.
- static void `CLOCK_ROOT_ControlByDomainMode` (`clock_root_t` name, `uint8_t` domainId)
Set this clock works in CPU Low Power Mode.
- static void `CLOCK_LPCG_LockControlMode` (`clock_lpcg_t` name)

- *Lock low power and access control mode for 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.*
static bool `CLOCK_LPCG_IsSetPointImplemented` (`clock_lpcg_t` name)
- *Check whether this clock implement SetPoint control scheme.*
static void `CLOCK_LPCG_ControlByUnassignedMode` (`clock_lpcg_t` name)
- *Set this clock works in Unassigned Mode.*
void `CLOCK_LPCG_ControlBySetPointMode` (`clock_lpcg_t` name, `uint16_t` spValue, `uint16_t` stbyValue)
- *Set this clock works in SetPoint control Mode.*
void `CLOCK_LPCG_ControlByCpuLowPowerMode` (`clock_lpcg_t` name, `uint8_t` domainId, `clock_level_t` level0, `clock_level_t` level1)
- *Set this clock works in CPU Low Power Mode.*
static void `CLOCK_LPCG_SetCurrentClockLevel` (`clock_lpcg_t` name, `clock_level_t` level)
- *Set clock depend level for current accessing domain.*
static void `CLOCK_LPCG_ControlByDomainMode` (`clock_lpcg_t` name, `uint8_t` domainId)
- *Set this clock works in Domain Mode.*

5.2 Data Structure Documentation

5.2.1 struct clock_group_config_t

Data Fields

- bool `clockOff`
Turn off the clock.
- `uint16_t` `resetDiv`
resetDiv + 1 should be common multiple of all dividers, valid range 0 ~ 255.
- `uint8_t` `div0`
Divide root clock by div0 + 1, valid range: 0 ~ 15.

Field Documentation

- (1) `bool clock_group_config_t::clockOff`
- (2) `uint16_t clock_group_config_t::resetDiv`
- (3) `uint8_t clock_group_config_t::div0`

5.2.2 struct clock_arm_pll_config_t

The output clock frequency is:

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

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.

5.2.3 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 - $F_{out} = F_{ref} * 20$; 1 - $F_{out} = F_{ref} * 22$

5.2.4 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](#)

5.2.5 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.

5.2.6 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

5.2.7 struct clock_video_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_video_pll_config_t::loopDivider`

Valid range for DIV_SELECT divider value: 27~54.

(2) `uint8_t clock_video_pll_config_t::postDivider`

(3) `uint32_t clock_video_pll_config_t::numerator`

5.2.8 struct clock_sys_pll1_gpc_config_t**Data Fields**

- `uint8_t loopDivider`
PLL loop divider.
- `uint32_t numerator`
30 bit numerator of fractional loop divider.
- `uint32_t denominator`
30 bit denominator of fractional loop divider
- `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_sys_pll1_gpc_config_t::loopDivider`

(2) `uint32_t clock_sys_pll1_gpc_config_t::numerator`

5.2.9 struct clock_enet_pll_config_t**Data Fields**

- `bool enableClkOutput`
Power on and enable PLL clock output for ENET0 (ref_enetpll0).
- `bool enableClkOutput25M`
Power on and enable PLL clock output for ENET2 (ref_enetpll2).
- `uint8_t loopDivider`
Controls the frequency of the ENET0 reference clock.
- `uint8_t src`
Pll clock source, reference _clock_pll_clk_src.
- `bool enableClkOutput1`
Power on and enable PLL clock output for ENET1 (ref_enetpll1).
- `uint8_t loopDivider1`
Controls the frequency of the ENET1 reference clock.

Field Documentation

(1) `bool clock_enet_pll_config_t::enableClkOutput`

(2) `bool clock_enet_pll_config_t::enableClkOutput25M`

(3) `uint8_t clock_enet_pll_config_t::loopDivider`

b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

(4) `bool clock_enet_pll_config_t::enableClkOutput1`

(5) `uint8_t clock_enet_pll_config_t::loopDivider1`

b00 25MHz b01 50MHz b10 100MHz (not 50% duty cycle) b11 125MHz

5.2.10 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`

5.2.11 struct clock_root_setpoint_config_t

Data Fields

- `uint8_t grade`
Indicate speed grade for each SetPoint.
- `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_setpoint_config_t::mux`

5.3 Macro Definition Documentation

5.3.1 #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.

5.3.2 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))

5.3.3 #define LPADC_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Lpadc1, \
    kCLOCK_Lpadc2 \
}
```

5.3.4 #define ADC_ETC_CLOCKS

Value:

```
{
    kCLOCK_Adc_Etc \
}
```

5.3.5 #define AOI_CLOCKS

Value:

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

5.3.6 #define DCDC_CLOCKS

Value:

```
{
    kCLOCK_Dcdc \
}
```

Clock ip name array for DCDC.

5.3.7 #define DCDC_CLOCKS

Value:

```
{  
    \kCLOCK_Dcdc \  
}
```

Clock ip name array for DCDC.

5.3.8 #define SRC_CLOCKS

Value:

```
{  
    \kCLOCK_Src \  
}
```

5.3.9 #define GPC_CLOCKS

Value:

```
{  
    \kCLOCK_Gpc \  
}
```

5.3.10 #define SSARC_CLOCKS

Value:

```
{  
    \kCLOCK_Ssarc \  
}
```

5.3.11 #define WDOG_CLOCKS

Value:

```
{  
    \kCLOCK_IpInvalid, kCLOCK_Wdog1,  
    kCLOCK_Wdog2, kCLOCK_Wdog3, kCLOCK_Wdog4 \  
}
```

5.3.12 #define EWM_CLOCKS

Value:

```
{  
    \kCLOCK_Ewm0 \  
}
```

5.3.13 #define SEMA_CLOCKS

Value:

```
{  
    \kCLOCK_Sema \  
}
```

5.3.14 #define MU_CLOCKS

Value:

```
{  
    \kCLOCK_Mu_B \  
}
```

5.3.15 #define EDMA_CLOCKS

Value:

```
{  
    \kCLOCK_Edma, kCLOCK_Edma_Lpsr \  
}
```

5.3.16 #define FLEXRAM_CLOCKS

Value:

```
{  
    \kCLOCK_Flexram \  
}
```

5.3.17 #define LMEM_CLOCKS

Value:

```
{
    kCLOCK_Lmem \
}
```

5.3.18 #define FLEXSPI_CLOCKS

Value:

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

5.3.19 #define RDC_CLOCKS

Value:

```
{
    kCLOCK_Rdc, kCLOCK_M7_Xrdc, \
    kCLOCK_M4_Xrdc \
}
```

5.3.20 #define SEMC_CLOCKS

Value:

```
{
    kCLOCK_Semc \
}
```

5.3.21 #define XECC_CLOCKS

Value:

```
{
    kCLOCK_Xecc \
}
```

5.3.22 #define IEE_CLOCKS

Value:

```
{  
    kCLOCK_Iee \  
}
```

5.3.23 #define KEYMANAGER_CLOCKS

Value:

```
{  
    kCLOCK_Key_Manager \  
}
```

5.3.24 #define PUF_CLOCKS

Value:

```
{  
    kCLOCK_Puf \  
}
```

5.3.25 #define OCOTP_CLOCKS

Value:

```
{  
    kCLOCK_Ocotp \  
}
```

5.3.26 #define CAAM_CLOCKS

Value:

```
{  
    kCLOCK_Caam \  
}
```

5.3.27 #define XBAR_CLOCKS

Value:

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

5.3.28 #define IOMUXC_CLOCKS

Value:

```
{
    kCLOCK_Iomuxc, kCLOCK_Iomuxc_Lpsr \
}
```

5.3.29 #define GPIO_CLOCKS

Value:

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Gpio, \
    kCLOCK_Gpio, kCLOCK_Gpio, kCLOCK_Gpio, \
    kCLOCK_Gpio, kCLOCK_Gpio, kCLOCK_Gpio, \
    kCLOCK_Gpio, kCLOCK_Gpio, \
    kCLOCK_Gpio, kCLOCK_Gpio, kCLOCK_Gpio, \
    kCLOCK_Gpio, \
}
```

5.3.30 #define KPP_CLOCKS

Value:

```
{
    kCLOCK_Kpp \
}
```

5.3.31 #define FLEXIO_CLOCKS

Value:

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

5.3.32 #define DAC_CLOCKS

Value:

```
{
    \
    kCLOCK_Dac \
}
```

5.3.33 #define CMP_CLOCKS

Value:

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Acmp1,
    kCLOCK_Acmp2, kCLOCK_Acmp3, kCLOCK_Acmp4 \
}
```

5.3.34 #define PIT_CLOCKS

Value:

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Pit1,
    kCLOCK_Pit2 \
}
```

5.3.35 #define GPT_CLOCKS

Value:

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

5.3.36 #define TMR_CLOCKS

Value:

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Qtimer1,
    kCLOCK_Qtimer2, kCLOCK_Qtimer3, kCLOCK_Qtimer4 \
}
```

5.3.37 #define ENC_CLOCKS

Value:

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

5.3.38 #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 \
    } \
}
```

5.3.39 #define FLEXCAN_CLOCKS

Value:

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

5.3.40 #define LPUART_CLOCKS

Value:

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


5.3.41 #define LPI2C_CLOCKS**Value:**

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Lpi2c1,
    kCLOCK_Lpi2c2, kCLOCK_Lpi2c3, kCLOCK_Lpi2c4,
    kCLOCK_Lpi2c5, kCLOCK_Lpi2c6 \
}
```

5.3.42 #define LPSPI_CLOCKS**Value:**

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Lpspi1,
    kCLOCK_Lpspi2, kCLOCK_Lpspi3, kCLOCK_Lpspi4,
    kCLOCK_Lpspi5, kCLOCK_Lpspi6 \
}
```

5.3.43 #define EMVSIM_CLOCKS**Value:**

```
{
    \
    kCLOCK_IpInvalid, kCLOCK_Sim1,
    kCLOCK_Sim2 \
}
```

5.3.44 #define ENET_CLOCKS**Value:**

```
{
    \
    kCLOCK_Enet, kCLOCK_Enet_1g \
}
```

5.3.45 #define ENETQOS_CLOCKS**Value:**

```
{
    \
    kCLOCK_Enet_Qos \
}
```

5.3.46 #define USB_CLOCKS

Value:

```
{  
    kCLOCK_Usb \  
}
```

5.3.47 #define CDOG_CLOCKS

Value:

```
{  
    kCLOCK_Cdog \  
}
```

5.3.48 #define USDHC_CLOCKS

Value:

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

5.3.49 #define ASRC_CLOCKS

Value:

```
{  
    kCLOCK_Asrc \  
}
```

5.3.50 #define MQS_CLOCKS

Value:

```
{  
    kCLOCK_Mqs \  
}
```

5.3.51 #define PDM_CLOCKS

Value:

```
{  
    \kCLOCK_Pdm \  
}
```

5.3.52 #define SPDIF_CLOCKS

Value:

```
{  
    \kCLOCK_Spdif \  
}
```

5.3.53 #define SAI_CLOCKS

Value:

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

5.3.54 #define PXP_CLOCKS

Value:

```
{  
    \kCLOCK_Pxp \  
}
```

5.3.55 #define GPU2D_CLOCKS

Value:

```
{  
    \kCLOCK_Gpu2d \  
}
```

5.3.56 #define LCDIF_CLOCKS

Value:

```
{  
    kCLOCK_Lcdif \  
}
```

5.3.57 #define LCDIFV2_CLOCKS

Value:

```
{  
    kCLOCK_Lcdifv2 \  
}
```

5.3.58 #define MIPI_DSI_HOST_CLOCKS

Value:

```
{  
    kCLOCK_Mipi_Dsi \  
}
```

5.3.59 #define MIPI_CSI2RX_CLOCKS

Value:

```
{  
    kCLOCK_Mipi_Csi \  
}
```

5.3.60 #define CSI_CLOCKS

Value:

```
{  
    kCLOCK_Csi \  
}
```

5.3.61 #define DCIC_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_Dcic_Mipi, \
    kCLOCK_Dcic_Lcd \
}
```

5.3.62 #define DMAMUX_CLOCKS

Value:

```
{
    kCLOCK_Edma, kCLOCK_Edma_Lpsr \
}
```

5.3.63 #define XBARA_CLOCKS

Value:

```
{
    kCLOCK_Xbar1 \
}
```

5.3.64 #define XBARB_CLOCKS

Value:

```
{
    kCLOCK_IpInvalid, kCLOCK_IpInvalid, \
    kCLOCK_Xbar2, kCLOCK_Xbar3 \
}
```

5.3.65 #define CLOCK_GetCoreSysClkFreq CLOCK_GetCpuClkFreq

5.4 Enumeration Type Documentation

5.4.1 enum clock_lpcg_t

Enumerator

kCLOCK_M7 Clock LPCG M7.

kCLOCK_M4 Clock LPCG M4.
kCLOCK_Sim_M7 Clock LPCG SIM M7.
kCLOCK_Sim_M Clock LPCG SIM M4.
kCLOCK_Sim_Disp Clock LPCG SIM DISP.
kCLOCK_Sim_Per Clock LPCG SIM PER.
kCLOCK_Sim_Lpsr Clock LPCG SIM LPSR.
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_Ssarc Clock LPCG SSARC.
kCLOCK_Sim_R Clock LPCG SIM_R.
kCLOCK_Wdog1 Clock LPCG WDOG1.
kCLOCK_Wdog2 Clock LPCG WDOG2.
kCLOCK_Wdog3 Clock LPCG WDOG3.
kCLOCK_Wdog4 Clock LPCG WDOG4.
kCLOCK_Ewm0 Clock LPCG EWM0.
kCLOCK_Sema Clock LPCG SEMA.
kCLOCK_Mu_A Clock LPCG MU_A.
kCLOCK_Mu_B Clock LPCG MU_B.
kCLOCK_Edma Clock LPCG EDMA.
kCLOCK_Edma_Lpsr Clock LPCG EDMA_LPSR.
kCLOCK_Romcp Clock LPCG ROMCP.
kCLOCK_Ocram Clock LPCG OCRAM.
kCLOCK_Flexram Clock LPCG FLEXRAM.
kCLOCK_Lmem Clock LPCG Lmem.
kCLOCK_Flexspi1 Clock LPCG Flexspi1.
kCLOCK_Flexspi2 Clock LPCG Flexspi2.
kCLOCK_Rdc Clock LPCG RDC.
kCLOCK_M7_Xrdc Clock LPCG M7 XRDC.
kCLOCK_M4_Xrdc Clock LPCG M4 XRDC.
kCLOCK_Semc Clock LPCG SEMC.
kCLOCK_Xecc Clock LPCG XECC.
kCLOCK_Iee Clock LPCG IEE.
kCLOCK_Key_Manager Clock LPCG KEY_MANAGER.
kCLOCK_Puf Clock LPCG PUF.
kCLOCK_Ocotp Clock LPCG OSOTP.
kCLOCK_Snvs_Hp Clock LPCG SNVS_HP.
kCLOCK_Snvs Clock LPCG SNVS.
kCLOCK_Caam Clock LPCG Caam.
kCLOCK_Jtag_Mux Clock LPCG JTAG_MUX.
kCLOCK_Cstrace Clock LPCG CSTRACE.
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_Adc_Etc Clock LPCG ADC_ETC.
kCLOCK_Iomuxc Clock LPCG IOMUXC.
kCLOCK_Iomuxc_Lpsr Clock LPCG IOMUXC_LPSR.
kCLOCK_Gpio Clock LPCG GPIO.
kCLOCK_Kpp Clock LPCG KPP.
kCLOCK_Flexio1 Clock LPCG FLEXIO1.
kCLOCK_Flexio2 Clock LPCG FLEXIO2.
kCLOCK_Lpadc1 Clock LPCG LPADC1.
kCLOCK_Lpadc2 Clock LPCG LPADC2.
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_Pit1 Clock LPCG PIT1.
kCLOCK_Pit2 Clock LPCG PIT2.
kCLOCK_Gpt1 Clock LPCG GPT1.
kCLOCK_Gpt2 Clock LPCG GPT2.
kCLOCK_Gpt3 Clock LPCG GPT3.
kCLOCK_Gpt4 Clock LPCG GPT4.
kCLOCK_Gpt5 Clock LPCG GPT5.
kCLOCK_Gpt6 Clock LPCG GPT6.
kCLOCK_Qtimer1 Clock LPCG QTIMER1.
kCLOCK_Qtimer2 Clock LPCG QTIMER2.
kCLOCK_Qtimer3 Clock LPCG QTIMER3.
kCLOCK_Qtimer4 Clock LPCG QTIMER4.
kCLOCK_Enc1 Clock LPCG Enc1.
kCLOCK_Enc2 Clock LPCG Enc2.
kCLOCK_Enc3 Clock LPCG Enc3.
kCLOCK_Enc4 Clock LPCG Enc4.
kCLOCK_Hrtimer Clock LPCG Hrtimer.
kCLOCK_Pwm1 Clock LPCG PWM1.
kCLOCK_Pwm2 Clock LPCG PWM2.
kCLOCK_Pwm3 Clock LPCG PWM3.
kCLOCK_Pwm4 Clock LPCG PWM4.
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_Sim1 Clock LPCG SIM1.
kCLOCK_Sim2 Clock LPCG SIM2.
kCLOCK_Enet Clock LPCG ENET.
kCLOCK_Enet_1g Clock LPCG ENET 1G.
kCLOCK_Enet_Qos Clock LPCG ENET QOS.
kCLOCK_Usb Clock LPCG USB.
kCLOCK_Cdog Clock LPCG CDOG.
kCLOCK_Usdhc1 Clock LPCG USDHC1.
kCLOCK_Usdhc2 Clock LPCG USDHC2.
kCLOCK_Asrc Clock LPCG ASRC.
kCLOCK_Mqs Clock LPCG MQS.
kCLOCK_Pdm Clock LPCG PDM.
kCLOCK_Spdif Clock LPCG SPDIF.
kCLOCK_Sai1 Clock LPCG SAI1.
kCLOCK_Sai2 Clock LPCG SAI2.
kCLOCK_Sai3 Clock LPCG SAI3.
kCLOCK_Sai4 Clock LPCG SAI4.
kCLOCK_Pxp Clock LPCG PXP.
kCLOCK_Gpu2d Clock LPCG GPU2D.
kCLOCK_Lcdif Clock LPCG LCDIF.
kCLOCK_Lcdifv2 Clock LPCG LCDIFV2.
kCLOCK_Mipi_Dsi Clock LPCG MIPI DSI.
kCLOCK_Mipi_Csi Clock LPCG MIPI CSI.
kCLOCK_Csi Clock LPCG CSI.
kCLOCK_Dcic_Mipi Clock LPCG DCIC MIPI.

kCLOCK_Dcic_Lcd Clock LPCG DCIC LCD.
kCLOCK_Video_Mux Clock LPCG VIDEO MUX.
kCLOCK_Uniq_Edt_I Clock LPCG Uniq_Edt_I.
kCLOCK_IpInvalid Invalid value.

5.4.2 enum clock_name_t

Enumerator

kCLOCK_OscRc16M 16MHz RC Oscillator.
kCLOCK_OscRc48M 48MHz RC Oscillator.
kCLOCK_OscRc48MDiv2 48MHz RC Oscillator Div2.
kCLOCK_OscRc400M 400MHz RC Oscillator.
kCLOCK_Osc24M 24MHz Oscillator.
kCLOCK_Osc24MOut 48MHz 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_VideoPll SYS VIDEO PLL.
kCLOCK_VideoPllOut SYS VIDEO PLL OUT.
kCLOCK_CpuClk SYS CPU CLK.
kCLOCK_CoreSysClk SYS CORE SYS CLK.

5.4.3 enum clock_root_t

Enumerator

kCLOCK_Root_M7 CLOCK Root M7.
kCLOCK_Root_M4 CLOCK Root M4.
kCLOCK_Root_Bus CLOCK Root Bus.
kCLOCK_Root_Bus_Lpsr CLOCK Root Bus Lpsr.
kCLOCK_Root_Semc CLOCK Root Semc.
kCLOCK_Root_Cssys CLOCK Root Cssys.
kCLOCK_Root_Cstrace CLOCK Root Cstrace.
kCLOCK_Root_M4_Systick CLOCK Root M4 Systick.
kCLOCK_Root_M7_Systick CLOCK Root M7 Systick.
kCLOCK_Root_Adc1 CLOCK Root Adc1.
kCLOCK_Root_Adc2 CLOCK Root Adc2.
kCLOCK_Root_Acmp CLOCK Root Acmp.
kCLOCK_Root_Flexio1 CLOCK Root Flexio1.
kCLOCK_Root_Flexio2 CLOCK Root Flexio2.
kCLOCK_Root_Gpt1 CLOCK Root Gpt1.
kCLOCK_Root_Gpt2 CLOCK Root Gpt2.
kCLOCK_Root_Gpt3 CLOCK Root Gpt3.
kCLOCK_Root_Gpt4 CLOCK Root Gpt4.
kCLOCK_Root_Gpt5 CLOCK Root Gpt5.
kCLOCK_Root_Gpt6 CLOCK Root Gpt6.
kCLOCK_Root_Flexspi1 CLOCK Root Flexspi1.
kCLOCK_Root_Flexspi2 CLOCK Root Flexspi2.
kCLOCK_Root_Can1 CLOCK Root Can1.
kCLOCK_Root_Can2 CLOCK Root Can2.
kCLOCK_Root_Can3 CLOCK Root Can3.
kCLOCK_Root_Lpuart1 CLOCK Root Lpuart1.
kCLOCK_Root_Lpuart2 CLOCK Root Lpuart2.
kCLOCK_Root_Lpuart3 CLOCK Root Lpuart3.
kCLOCK_Root_Lpuart4 CLOCK Root Lpuart4.
kCLOCK_Root_Lpuart5 CLOCK Root Lpuart5.
kCLOCK_Root_Lpuart6 CLOCK Root Lpuart6.
kCLOCK_Root_Lpuart7 CLOCK Root Lpuart7.
kCLOCK_Root_Lpuart8 CLOCK Root Lpuart8.
kCLOCK_Root_Lpuart9 CLOCK Root Lpuart9.
kCLOCK_Root_Lpuart10 CLOCK Root Lpuart10.
kCLOCK_Root_Lpuart11 CLOCK Root Lpuart11.
kCLOCK_Root_Lpuart12 CLOCK Root Lpuart12.
kCLOCK_Root_Lpi2c1 CLOCK Root Lpi2c1.
kCLOCK_Root_Lpi2c2 CLOCK Root Lpi2c2.
kCLOCK_Root_Lpi2c3 CLOCK Root Lpi2c3.
kCLOCK_Root_Lpi2c4 CLOCK Root Lpi2c4.

kCLOCK_Root_Lpi2c5 CLOCK Root Lpi2c5.
kCLOCK_Root_Lpi2c6 CLOCK Root Lpi2c6.
kCLOCK_Root_Lpspi1 CLOCK Root Lpspi1.
kCLOCK_Root_Lpspi2 CLOCK Root Lpspi2.
kCLOCK_Root_Lpspi3 CLOCK Root Lpspi3.
kCLOCK_Root_Lpspi4 CLOCK Root Lpspi4.
kCLOCK_Root_Lpspi5 CLOCK Root Lpspi5.
kCLOCK_Root_Lpspi6 CLOCK Root Lpspi6.
kCLOCK_Root_Emv1 CLOCK Root Emv1.
kCLOCK_Root_Emv2 CLOCK Root Emv2.
kCLOCK_Root_Enet1 CLOCK Root Enet1.
kCLOCK_Root_Enet2 CLOCK Root Enet2.
kCLOCK_Root_Enet_Qos CLOCK Root Enet Qos.
kCLOCK_Root_Enet_25m CLOCK Root Enet 25M.
kCLOCK_Root_Enet_Timer1 CLOCK Root Enet Timer1.
kCLOCK_Root_Enet_Timer2 CLOCK Root Enet Timer2.
kCLOCK_Root_Enet_Timer3 CLOCK Root Enet Timer3.
kCLOCK_Root_Usdhc1 CLOCK Root Usdhc1.
kCLOCK_Root_Usdhc2 CLOCK Root Usdhc2.
kCLOCK_Root_Asrc CLOCK Root Asrc.
kCLOCK_Root_Mqs CLOCK Root Mqs.
kCLOCK_Root_Mic CLOCK Root MIC.
kCLOCK_Root_Spdif CLOCK Root Spdif.
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_Gc355 CLOCK Root Gc355.
kCLOCK_Root_Lcdif CLOCK Root Lcdif.
kCLOCK_Root_Lcdifv2 CLOCK Root Lcdifv2.
kCLOCK_Root_Mipi_Ref CLOCK Root Mipi Ref.
kCLOCK_Root_Mipi_Esc CLOCK Root Mipi Esc.
kCLOCK_Root_Csi2 CLOCK Root Csi2.
kCLOCK_Root_Csi2_Esc CLOCK Root Csi2 Esc.
kCLOCK_Root_Csi2_Ui CLOCK Root Csi2 Ui.
kCLOCK_Root_Csi CLOCK Root Csi.
kCLOCK_Root_Cko1 CLOCK Root CKo1.
kCLOCK_Root_Cko2 CLOCK Root CKo2.

5.4.4 enum clock_root_mux_source_t

Enumerator

kCLOCK_M7_ClockRoot_MuxOscRc48MDiv2 M7 mux from MuxOscRc48MDiv2.

kCLOCK_M7_ClockRoot_MuxOsc24MOut M7 mux from MuxOsc24MOut.
kCLOCK_M7_ClockRoot_MuxOscRc400M M7 mux from MuxOscRc400M.
kCLOCK_M7_ClockRoot_MuxOscRc16M M7 mux from MuxOscRc16M.
kCLOCK_M7_ClockRoot_MuxArmPllOut M7 mux from MuxArmPllOut.
kCLOCK_M7_ClockRoot_MuxSysPll1Out M7 mux from MuxSysPll1Out.
kCLOCK_M7_ClockRoot_MuxSysPll3Out M7 mux from MuxSysPll3Out.
kCLOCK_M7_ClockRoot_MuxVideoPllOut M7 mux from MuxVideoPllOut.
kCLOCK_M4_ClockRoot_MuxOscRc48MDiv2 M4 mux from MuxOscRc48MDiv2.
kCLOCK_M4_ClockRoot_MuxOsc24MOut M4 mux from MuxOsc24MOut.
kCLOCK_M4_ClockRoot_MuxOscRc400M M4 mux from MuxOscRc400M.
kCLOCK_M4_ClockRoot_MuxOscRc16M M4 mux from MuxOscRc16M.
kCLOCK_M4_ClockRoot_MuxSysPll3Pfd3 M4 mux from MuxSysPll3Pfd3.
kCLOCK_M4_ClockRoot_MuxSysPll3Out M4 mux from MuxSysPll3Out.
kCLOCK_M4_ClockRoot_MuxSysPll2Out M4 mux from MuxSysPll2Out.
kCLOCK_M4_ClockRoot_MuxSysPll1Div5 M4 mux from MuxSysPll1Div5.
kCLOCK_BUS_ClockRoot_MuxOscRc48MDiv2 BUS mux from MuxOscRc48MDiv2.
kCLOCK_BUS_ClockRoot_MuxOsc24MOut BUS mux from MuxOsc24MOut.
kCLOCK_BUS_ClockRoot_MuxOscRc400M BUS mux from MuxOscRc400M.
kCLOCK_BUS_ClockRoot_MuxOscRc16M BUS mux from MuxOscRc16M.
kCLOCK_BUS_ClockRoot_MuxSysPll3Out BUS mux from MuxSysPll3Out.
kCLOCK_BUS_ClockRoot_MuxSysPll1Div5 BUS mux from MuxSysPll1Div5.
kCLOCK_BUS_ClockRoot_MuxSysPll2Out BUS mux from MuxSysPll2Out.
kCLOCK_BUS_ClockRoot_MuxSysPll2Pfd3 BUS mux from MuxSysPll2Pfd3.
kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc48MDiv2 BUS_LPSR mux from MuxOscRc48M-Div2.
kCLOCK_BUS_LPSR_ClockRoot_MuxOsc24MOut BUS_LPSR mux from MuxOsc24MOut.
kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc400M BUS_LPSR mux from MuxOscRc400M.
kCLOCK_BUS_LPSR_ClockRoot_MuxOscRc16M BUS_LPSR mux from MuxOscRc16M.
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll3Pfd3 BUS_LPSR mux from MuxSysPll3Pfd3.
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll3Out BUS_LPSR mux from MuxSysPll3Out.
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll2Out BUS_LPSR mux from MuxSysPll2Out.
kCLOCK_BUS_LPSR_ClockRoot_MuxSysPll1Div5 BUS_LPSR mux from MuxSysPll1Div5.
kCLOCK_SEMC_ClockRoot_MuxOscRc48MDiv2 SEMC mux from MuxOscRc48MDiv2.
kCLOCK_SEMC_ClockRoot_MuxOsc24MOut SEMC mux from MuxOsc24MOut.
kCLOCK_SEMC_ClockRoot_MuxOscRc400M SEMC mux from MuxOscRc400M.
kCLOCK_SEMC_ClockRoot_MuxOscRc16M SEMC mux from MuxOscRc16M.
kCLOCK_SEMC_ClockRoot_MuxSysPll1Div5 SEMC mux from MuxSysPll1Div5.
kCLOCK_SEMC_ClockRoot_MuxSysPll2Out SEMC mux from MuxSysPll2Out.
kCLOCK_SEMC_ClockRoot_MuxSysPll2Pfd1 SEMC mux from MuxSysPll2Pfd1.
kCLOCK_SEMC_ClockRoot_MuxSysPll3Pfd0 SEMC mux from MuxSysPll3Pfd0.
kCLOCK_CSSYS_ClockRoot_MuxOscRc48MDiv2 CSSYS mux from MuxOscRc48MDiv2.
kCLOCK_CSSYS_ClockRoot_MuxOsc24MOut CSSYS mux from MuxOsc24MOut.
kCLOCK_CSSYS_ClockRoot_MuxOscRc400M CSSYS mux from MuxOscRc400M.
kCLOCK_CSSYS_ClockRoot_MuxOscRc16M CSSYS mux from MuxOscRc16M.
kCLOCK_CSSYS_ClockRoot_MuxSysPll3Div2 CSSYS mux from MuxSysPll3Div2.

kCLOCK_CSSYS_ClockRoot_MuxSysPll1Div5 CSSYS mux from MuxSysPll1Div5.
kCLOCK_CSSYS_ClockRoot_MuxSysPll2Out CSSYS mux from MuxSysPll2Out.
kCLOCK_CSSYS_ClockRoot_MuxSysPll2Pfd3 CSSYS mux from MuxSysPll2Pfd3.
kCLOCK_CSTRACE_ClockRoot_MuxOscRc48MDiv2 CSTRACE mux from MuxOscRc48M-Div2.
kCLOCK_CSTRACE_ClockRoot_MuxOsc24MOut CSTRACE mux from MuxOsc24MOut.
kCLOCK_CSTRACE_ClockRoot_MuxOscRc400M CSTRACE mux from MuxOscRc400M.
kCLOCK_CSTRACE_ClockRoot_MuxOscRc16M CSTRACE mux from MuxOscRc16M.
kCLOCK_CSTRACE_ClockRoot_MuxSysPll3Div2 CSTRACE mux from MuxSysPll3Div2.
kCLOCK_CSTRACE_ClockRoot_MuxSysPll1Div5 CSTRACE mux from MuxSysPll1Div5.
kCLOCK_CSTRACE_ClockRoot_MuxSysPll2Pfd1 CSTRACE mux from MuxSysPll2Pfd1.
kCLOCK_CSTRACE_ClockRoot_MuxSysPll2Out CSTRACE mux from MuxSysPll2Out.
kCLOCK_M4_SYSTICK_ClockRoot_MuxOscRc48MDiv2 M4_SYSTICK mux from MuxOsc-Rc48MDiv2.
kCLOCK_M4_SYSTICK_ClockRoot_MuxOsc24MOut M4_SYSTICK mux from MuxOsc24M-Out.
kCLOCK_M4_SYSTICK_ClockRoot_MuxOscRc400M M4_SYSTICK mux from MuxOscRc400-M.
kCLOCK_M4_SYSTICK_ClockRoot_MuxOscRc16M M4_SYSTICK mux from MuxOscRc16M.
kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll3Pfd3 M4_SYSTICK mux from MuxSysPll3-Pfd3.
kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll3Out M4_SYSTICK mux from MuxSysPll3Out.
kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll2Pfd0 M4_SYSTICK mux from MuxSysPll2-Pfd0.
kCLOCK_M4_SYSTICK_ClockRoot_MuxSysPll1Div5 M4_SYSTICK mux from MuxSysPll1-Div5.
kCLOCK_M7_SYSTICK_ClockRoot_MuxOscRc48MDiv2 M7_SYSTICK mux from MuxOsc-Rc48MDiv2.
kCLOCK_M7_SYSTICK_ClockRoot_MuxOsc24MOut M7_SYSTICK mux from MuxOsc24M-Out.
kCLOCK_M7_SYSTICK_ClockRoot_MuxOscRc400M M7_SYSTICK mux from MuxOscRc400-M.
kCLOCK_M7_SYSTICK_ClockRoot_MuxOscRc16M M7_SYSTICK mux from MuxOscRc16M.
kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll2Out M7_SYSTICK mux from MuxSysPll2Out.
kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll3Div2 M7_SYSTICK mux from MuxSysPll3-Div2.
kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll1Div5 M7_SYSTICK mux from MuxSysPll1-Div5.
kCLOCK_M7_SYSTICK_ClockRoot_MuxSysPll2Pfd0 M7_SYSTICK mux from MuxSysPll2-Pfd0.
kCLOCK_ADC1_ClockRoot_MuxOscRc48MDiv2 ADC1 mux from MuxOscRc48MDiv2.
kCLOCK_ADC1_ClockRoot_MuxOsc24MOut ADC1 mux from MuxOsc24MOut.
kCLOCK_ADC1_ClockRoot_MuxOscRc400M ADC1 mux from MuxOscRc400M.
kCLOCK_ADC1_ClockRoot_MuxOscRc16M ADC1 mux from MuxOscRc16M.
kCLOCK_ADC1_ClockRoot_MuxSysPll3Div2 ADC1 mux from MuxSysPll3Div2.

kCLOCK_ADC1_ClockRoot_MuxSysPll1Div5 ADC1 mux from MuxSysPll1Div5.
kCLOCK_ADC1_ClockRoot_MuxSysPll2Out ADC1 mux from MuxSysPll2Out.
kCLOCK_ADC1_ClockRoot_MuxSysPll2Pfd3 ADC1 mux from MuxSysPll2Pfd3.
kCLOCK_ADC2_ClockRoot_MuxOscRc48MDiv2 ADC2 mux from MuxOscRc48MDiv2.
kCLOCK_ADC2_ClockRoot_MuxOsc24MOut ADC2 mux from MuxOsc24MOut.
kCLOCK_ADC2_ClockRoot_MuxOscRc400M ADC2 mux from MuxOscRc400M.
kCLOCK_ADC2_ClockRoot_MuxOscRc16M ADC2 mux from MuxOscRc16M.
kCLOCK_ADC2_ClockRoot_MuxSysPll3Div2 ADC2 mux from MuxSysPll3Div2.
kCLOCK_ADC2_ClockRoot_MuxSysPll1Div5 ADC2 mux from MuxSysPll1Div5.
kCLOCK_ADC2_ClockRoot_MuxSysPll2Out ADC2 mux from MuxSysPll2Out.
kCLOCK_ADC2_ClockRoot_MuxSysPll2Pfd3 ADC2 mux from MuxSysPll2Pfd3.
kCLOCK_ACMP_ClockRoot_MuxOscRc48MDiv2 ACMP mux from MuxOscRc48MDiv2.
kCLOCK_ACMP_ClockRoot_MuxOsc24MOut ACMP mux from MuxOsc24MOut.
kCLOCK_ACMP_ClockRoot_MuxOscRc400M ACMP mux from MuxOscRc400M.
kCLOCK_ACMP_ClockRoot_MuxOscRc16M ACMP mux from MuxOscRc16M.
kCLOCK_ACMP_ClockRoot_MuxSysPll3Out ACMP mux from MuxSysPll3Out.
kCLOCK_ACMP_ClockRoot_MuxSysPll1Div5 ACMP mux from MuxSysPll1Div5.
kCLOCK_ACMP_ClockRoot_MuxAudioPllOut ACMP mux from MuxAudioPllOut.
kCLOCK_ACMP_ClockRoot_MuxSysPll2Pfd3 ACMP mux from MuxSysPll2Pfd3.
kCLOCK_FLEXIO1_ClockRoot_MuxOscRc48MDiv2 FLEXIO1 mux from MuxOscRc48MDiv2.

kCLOCK_FLEXIO1_ClockRoot_MuxOsc24MOut FLEXIO1 mux from MuxOsc24MOut.
kCLOCK_FLEXIO1_ClockRoot_MuxOscRc400M FLEXIO1 mux from MuxOscRc400M.
kCLOCK_FLEXIO1_ClockRoot_MuxOscRc16M FLEXIO1 mux from MuxOscRc16M.
kCLOCK_FLEXIO1_ClockRoot_MuxSysPll3Div2 FLEXIO1 mux from MuxSysPll3Div2.
kCLOCK_FLEXIO1_ClockRoot_MuxSysPll1Div5 FLEXIO1 mux from MuxSysPll1Div5.
kCLOCK_FLEXIO1_ClockRoot_MuxSysPll2Out FLEXIO1 mux from MuxSysPll2Out.
kCLOCK_FLEXIO1_ClockRoot_MuxSysPll2Pfd3 FLEXIO1 mux from MuxSysPll2Pfd3.
kCLOCK_FLEXIO2_ClockRoot_MuxOscRc48MDiv2 FLEXIO2 mux from MuxOscRc48MDiv2.

kCLOCK_FLEXIO2_ClockRoot_MuxOsc24MOut FLEXIO2 mux from MuxOsc24MOut.
kCLOCK_FLEXIO2_ClockRoot_MuxOscRc400M FLEXIO2 mux from MuxOscRc400M.
kCLOCK_FLEXIO2_ClockRoot_MuxOscRc16M FLEXIO2 mux from MuxOscRc16M.
kCLOCK_FLEXIO2_ClockRoot_MuxSysPll3Div2 FLEXIO2 mux from MuxSysPll3Div2.
kCLOCK_FLEXIO2_ClockRoot_MuxSysPll1Div5 FLEXIO2 mux from MuxSysPll1Div5.
kCLOCK_FLEXIO2_ClockRoot_MuxSysPll2Out FLEXIO2 mux from MuxSysPll2Out.
kCLOCK_FLEXIO2_ClockRoot_MuxSysPll2Pfd3 FLEXIO2 mux from MuxSysPll2Pfd3.
kCLOCK_GPT1_ClockRoot_MuxOscRc48MDiv2 GPT1 mux from MuxOscRc48MDiv2.
kCLOCK_GPT1_ClockRoot_MuxOsc24MOut GPT1 mux from MuxOsc24MOut.
kCLOCK_GPT1_ClockRoot_MuxOscRc400M GPT1 mux from MuxOscRc400M.
kCLOCK_GPT1_ClockRoot_MuxOscRc16M GPT1 mux from MuxOscRc16M.
kCLOCK_GPT1_ClockRoot_MuxSysPll3Div2 GPT1 mux from MuxSysPll3Div2.
kCLOCK_GPT1_ClockRoot_MuxSysPll1Div5 GPT1 mux from MuxSysPll1Div5.
kCLOCK_GPT1_ClockRoot_MuxSysPll3Pfd2 GPT1 mux from MuxSysPll3Pfd2.
kCLOCK_GPT1_ClockRoot_MuxSysPll3Pfd3 GPT1 mux from MuxSysPll3Pfd3.

kCLOCK_GPT2_ClockRoot_MuxOscRc48MDiv2 GPT2 mux from MuxOscRc48MDiv2.
kCLOCK_GPT2_ClockRoot_MuxOsc24MOut GPT2 mux from MuxOsc24MOut.
kCLOCK_GPT2_ClockRoot_MuxOscRc400M GPT2 mux from MuxOscRc400M.
kCLOCK_GPT2_ClockRoot_MuxOscRc16M GPT2 mux from MuxOscRc16M.
kCLOCK_GPT2_ClockRoot_MuxSysPll3Div2 GPT2 mux from MuxSysPll3Div2.
kCLOCK_GPT2_ClockRoot_MuxSysPll1Div5 GPT2 mux from MuxSysPll1Div5.
kCLOCK_GPT2_ClockRoot_MuxAudioPllOut GPT2 mux from MuxAudioPllOut.
kCLOCK_GPT2_ClockRoot_MuxVideoPllOut GPT2 mux from MuxVideoPllOut.
kCLOCK_GPT3_ClockRoot_MuxOscRc48MDiv2 GPT3 mux from MuxOscRc48MDiv2.
kCLOCK_GPT3_ClockRoot_MuxOsc24MOut GPT3 mux from MuxOsc24MOut.
kCLOCK_GPT3_ClockRoot_MuxOscRc400M GPT3 mux from MuxOscRc400M.
kCLOCK_GPT3_ClockRoot_MuxOscRc16M GPT3 mux from MuxOscRc16M.
kCLOCK_GPT3_ClockRoot_MuxSysPll3Div2 GPT3 mux from MuxSysPll3Div2.
kCLOCK_GPT3_ClockRoot_MuxSysPll1Div5 GPT3 mux from MuxSysPll1Div5.
kCLOCK_GPT3_ClockRoot_MuxAudioPllOut GPT3 mux from MuxAudioPllOut.
kCLOCK_GPT3_ClockRoot_MuxVideoPllOut GPT3 mux from MuxVideoPllOut.
kCLOCK_GPT4_ClockRoot_MuxOscRc48MDiv2 GPT4 mux from MuxOscRc48MDiv2.
kCLOCK_GPT4_ClockRoot_MuxOsc24MOut GPT4 mux from MuxOsc24MOut.
kCLOCK_GPT4_ClockRoot_MuxOscRc400M GPT4 mux from MuxOscRc400M.
kCLOCK_GPT4_ClockRoot_MuxOscRc16M GPT4 mux from MuxOscRc16M.
kCLOCK_GPT4_ClockRoot_MuxSysPll3Div2 GPT4 mux from MuxSysPll3Div2.
kCLOCK_GPT4_ClockRoot_MuxSysPll1Div5 GPT4 mux from MuxSysPll1Div5.
kCLOCK_GPT4_ClockRoot_MuxSysPll3Pfd2 GPT4 mux from MuxSysPll3Pfd2.
kCLOCK_GPT4_ClockRoot_MuxSysPll3Pfd3 GPT4 mux from MuxSysPll3Pfd3.
kCLOCK_GPT5_ClockRoot_MuxOscRc48MDiv2 GPT5 mux from MuxOscRc48MDiv2.
kCLOCK_GPT5_ClockRoot_MuxOsc24MOut GPT5 mux from MuxOsc24MOut.
kCLOCK_GPT5_ClockRoot_MuxOscRc400M GPT5 mux from MuxOscRc400M.
kCLOCK_GPT5_ClockRoot_MuxOscRc16M GPT5 mux from MuxOscRc16M.
kCLOCK_GPT5_ClockRoot_MuxSysPll3Div2 GPT5 mux from MuxSysPll3Div2.
kCLOCK_GPT5_ClockRoot_MuxSysPll1Div5 GPT5 mux from MuxSysPll1Div5.
kCLOCK_GPT5_ClockRoot_MuxSysPll3Pfd2 GPT5 mux from MuxSysPll3Pfd2.
kCLOCK_GPT5_ClockRoot_MuxSysPll3Pfd3 GPT5 mux from MuxSysPll3Pfd3.
kCLOCK_GPT6_ClockRoot_MuxOscRc48MDiv2 GPT6 mux from MuxOscRc48MDiv2.
kCLOCK_GPT6_ClockRoot_MuxOsc24MOut GPT6 mux from MuxOsc24MOut.
kCLOCK_GPT6_ClockRoot_MuxOscRc400M GPT6 mux from MuxOscRc400M.
kCLOCK_GPT6_ClockRoot_MuxOscRc16M GPT6 mux from MuxOscRc16M.
kCLOCK_GPT6_ClockRoot_MuxSysPll3Div2 GPT6 mux from MuxSysPll3Div2.
kCLOCK_GPT6_ClockRoot_MuxSysPll1Div5 GPT6 mux from MuxSysPll1Div5.
kCLOCK_GPT6_ClockRoot_MuxSysPll3Pfd2 GPT6 mux from MuxSysPll3Pfd2.
kCLOCK_GPT6_ClockRoot_MuxSysPll3Pfd3 GPT6 mux from MuxSysPll3Pfd3.
kCLOCK_FLEXSPI1_ClockRoot_MuxOscRc48MDiv2 FLEXSPI1 mux from MuxOscRc48M-Div2.
kCLOCK_FLEXSPI1_ClockRoot_MuxOsc24MOut FLEXSPI1 mux from MuxOsc24MOut.
kCLOCK_FLEXSPI1_ClockRoot_MuxOscRc400M FLEXSPI1 mux from MuxOscRc400M.
kCLOCK_FLEXSPI1_ClockRoot_MuxOscRc16M FLEXSPI1 mux from MuxOscRc16M.

kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll3Pfd0 FLEXSPI1 mux from MuxSysPll3Pfd0.

kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll2Out FLEXSPI1 mux from MuxSysPll2Out.

kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll2Pfd2 FLEXSPI1 mux from MuxSysPll2Pfd2.

kCLOCK_FLEXSPI1_ClockRoot_MuxSysPll3Out FLEXSPI1 mux from MuxSysPll3Out.

kCLOCK_FLEXSPI2_ClockRoot_MuxOscRc48MDiv2 FLEXSPI2 mux from MuxOscRc48M-Div2.

kCLOCK_FLEXSPI2_ClockRoot_MuxOsc24MOut FLEXSPI2 mux from MuxOsc24MOut.

kCLOCK_FLEXSPI2_ClockRoot_MuxOscRc400M FLEXSPI2 mux from MuxOscRc400M.

kCLOCK_FLEXSPI2_ClockRoot_MuxOscRc16M FLEXSPI2 mux from MuxOscRc16M.

kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll3Pfd0 FLEXSPI2 mux from MuxSysPll3Pfd0.

kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll2Out FLEXSPI2 mux from MuxSysPll2Out.

kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll2Pfd2 FLEXSPI2 mux from MuxSysPll2Pfd2.

kCLOCK_FLEXSPI2_ClockRoot_MuxSysPll3Out FLEXSPI2 mux from MuxSysPll3Out.

kCLOCK_CAN1_ClockRoot_MuxOscRc48MDiv2 CAN1 mux from MuxOscRc48MDiv2.

kCLOCK_CAN1_ClockRoot_MuxOsc24MOut CAN1 mux from MuxOsc24MOut.

kCLOCK_CAN1_ClockRoot_MuxOscRc400M CAN1 mux from MuxOscRc400M.

kCLOCK_CAN1_ClockRoot_MuxOscRc16M CAN1 mux from MuxOscRc16M.

kCLOCK_CAN1_ClockRoot_MuxSysPll3Div2 CAN1 mux from MuxSysPll3Div2.

kCLOCK_CAN1_ClockRoot_MuxSysPll1Div5 CAN1 mux from MuxSysPll1Div5.

kCLOCK_CAN1_ClockRoot_MuxSysPll2Out CAN1 mux from MuxSysPll2Out.

kCLOCK_CAN1_ClockRoot_MuxSysPll2Pfd3 CAN1 mux from MuxSysPll2Pfd3.

kCLOCK_CAN2_ClockRoot_MuxOscRc48MDiv2 CAN2 mux from MuxOscRc48MDiv2.

kCLOCK_CAN2_ClockRoot_MuxOsc24MOut CAN2 mux from MuxOsc24MOut.

kCLOCK_CAN2_ClockRoot_MuxOscRc400M CAN2 mux from MuxOscRc400M.

kCLOCK_CAN2_ClockRoot_MuxOscRc16M CAN2 mux from MuxOscRc16M.

kCLOCK_CAN2_ClockRoot_MuxSysPll3Div2 CAN2 mux from MuxSysPll3Div2.

kCLOCK_CAN2_ClockRoot_MuxSysPll1Div5 CAN2 mux from MuxSysPll1Div5.

kCLOCK_CAN2_ClockRoot_MuxSysPll2Out CAN2 mux from MuxSysPll2Out.

kCLOCK_CAN2_ClockRoot_MuxSysPll2Pfd3 CAN2 mux from MuxSysPll2Pfd3.

kCLOCK_CAN3_ClockRoot_MuxOscRc48MDiv2 CAN3 mux from MuxOscRc48MDiv2.

kCLOCK_CAN3_ClockRoot_MuxOsc24MOut CAN3 mux from MuxOsc24MOut.

kCLOCK_CAN3_ClockRoot_MuxOscRc400M CAN3 mux from MuxOscRc400M.

kCLOCK_CAN3_ClockRoot_MuxOscRc16M CAN3 mux from MuxOscRc16M.

kCLOCK_CAN3_ClockRoot_MuxSysPll3Pfd3 CAN3 mux from MuxSysPll3Pfd3.

kCLOCK_CAN3_ClockRoot_MuxSysPll3Out CAN3 mux from MuxSysPll3Out.

kCLOCK_CAN3_ClockRoot_MuxSysPll2Pfd3 CAN3 mux from MuxSysPll2Pfd3.

kCLOCK_CAN3_ClockRoot_MuxSysPll1Div5 CAN3 mux from MuxSysPll1Div5.

kCLOCK_LPUART1_ClockRoot_MuxOscRc48MDiv2 LPUART1 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART1_ClockRoot_MuxOsc24MOut LPUART1 mux from MuxOsc24MOut.

kCLOCK_LPUART1_ClockRoot_MuxOscRc400M LPUART1 mux from MuxOscRc400M.

kCLOCK_LPUART1_ClockRoot_MuxOscRc16M LPUART1 mux from MuxOscRc16M.

kCLOCK_LPUART1_ClockRoot_MuxSysPll3Div2 LPUART1 mux from MuxSysPll3Div2.

kCLOCK_LPUART1_ClockRoot_MuxSysPll1Div5 LPUART1 mux from MuxSysPll1Div5.

kCLOCK_LPUART1_ClockRoot_MuxSysPll2Out LPUART1 mux from MuxSysPll2Out.

kCLOCK_LPUART1_ClockRoot_MuxSysPll2Pfd3 LPUART1 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART2_ClockRoot_MuxOscRc48MDiv2 LPUART2 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART2_ClockRoot_MuxOsc24MOut LPUART2 mux from MuxOsc24MOut.

kCLOCK_LPUART2_ClockRoot_MuxOscRc400M LPUART2 mux from MuxOscRc400M.

kCLOCK_LPUART2_ClockRoot_MuxOscRc16M LPUART2 mux from MuxOscRc16M.

kCLOCK_LPUART2_ClockRoot_MuxSysPll3Div2 LPUART2 mux from MuxSysPll3Div2.

kCLOCK_LPUART2_ClockRoot_MuxSysPll1Div5 LPUART2 mux from MuxSysPll1Div5.

kCLOCK_LPUART2_ClockRoot_MuxSysPll2Out LPUART2 mux from MuxSysPll2Out.

kCLOCK_LPUART2_ClockRoot_MuxSysPll2Pfd3 LPUART2 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART3_ClockRoot_MuxOscRc48MDiv2 LPUART3 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART3_ClockRoot_MuxOsc24MOut LPUART3 mux from MuxOsc24MOut.

kCLOCK_LPUART3_ClockRoot_MuxOscRc400M LPUART3 mux from MuxOscRc400M.

kCLOCK_LPUART3_ClockRoot_MuxOscRc16M LPUART3 mux from MuxOscRc16M.

kCLOCK_LPUART3_ClockRoot_MuxSysPll3Div2 LPUART3 mux from MuxSysPll3Div2.

kCLOCK_LPUART3_ClockRoot_MuxSysPll1Div5 LPUART3 mux from MuxSysPll1Div5.

kCLOCK_LPUART3_ClockRoot_MuxSysPll2Out LPUART3 mux from MuxSysPll2Out.

kCLOCK_LPUART3_ClockRoot_MuxSysPll2Pfd3 LPUART3 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART4_ClockRoot_MuxOscRc48MDiv2 LPUART4 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART4_ClockRoot_MuxOsc24MOut LPUART4 mux from MuxOsc24MOut.

kCLOCK_LPUART4_ClockRoot_MuxOscRc400M LPUART4 mux from MuxOscRc400M.

kCLOCK_LPUART4_ClockRoot_MuxOscRc16M LPUART4 mux from MuxOscRc16M.

kCLOCK_LPUART4_ClockRoot_MuxSysPll3Div2 LPUART4 mux from MuxSysPll3Div2.

kCLOCK_LPUART4_ClockRoot_MuxSysPll1Div5 LPUART4 mux from MuxSysPll1Div5.

kCLOCK_LPUART4_ClockRoot_MuxSysPll2Out LPUART4 mux from MuxSysPll2Out.

kCLOCK_LPUART4_ClockRoot_MuxSysPll2Pfd3 LPUART4 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART5_ClockRoot_MuxOscRc48MDiv2 LPUART5 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART5_ClockRoot_MuxOsc24MOut LPUART5 mux from MuxOsc24MOut.

kCLOCK_LPUART5_ClockRoot_MuxOscRc400M LPUART5 mux from MuxOscRc400M.

kCLOCK_LPUART5_ClockRoot_MuxOscRc16M LPUART5 mux from MuxOscRc16M.

kCLOCK_LPUART5_ClockRoot_MuxSysPll3Div2 LPUART5 mux from MuxSysPll3Div2.

kCLOCK_LPUART5_ClockRoot_MuxSysPll1Div5 LPUART5 mux from MuxSysPll1Div5.

kCLOCK_LPUART5_ClockRoot_MuxSysPll2Out LPUART5 mux from MuxSysPll2Out.

kCLOCK_LPUART5_ClockRoot_MuxSysPll2Pfd3 LPUART5 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART6_ClockRoot_MuxOscRc48MDiv2 LPUART6 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART6_ClockRoot_MuxOsc24MOut LPUART6 mux from MuxOsc24MOut.

kCLOCK_LPUART6_ClockRoot_MuxOscRc400M LPUART6 mux from MuxOscRc400M.

kCLOCK_LPUART6_ClockRoot_MuxOscRc16M LPUART6 mux from MuxOscRc16M.

kCLOCK_LPUART6_ClockRoot_MuxSysPll3Div2 LPUART6 mux from MuxSysPll3Div2.

kCLOCK_LPUART6_ClockRoot_MuxSysPll1Div5 LPUART6 mux from MuxSysPll1Div5.

kCLOCK_LPUART6_ClockRoot_MuxSysPll2Out LPUART6 mux from MuxSysPll2Out.

kCLOCK_LPUART6_ClockRoot_MuxSysPll2Pfd3 LPUART6 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART7_ClockRoot_MuxOscRc48MDiv2 LPUART7 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART7_ClockRoot_MuxOsc24MOut LPUART7 mux from MuxOsc24MOut.

kCLOCK_LPUART7_ClockRoot_MuxOscRc400M LPUART7 mux from MuxOscRc400M.

kCLOCK_LPUART7_ClockRoot_MuxOscRc16M LPUART7 mux from MuxOscRc16M.

kCLOCK_LPUART7_ClockRoot_MuxSysPll3Div2 LPUART7 mux from MuxSysPll3Div2.

kCLOCK_LPUART7_ClockRoot_MuxSysPll1Div5 LPUART7 mux from MuxSysPll1Div5.

kCLOCK_LPUART7_ClockRoot_MuxSysPll2Out LPUART7 mux from MuxSysPll2Out.

kCLOCK_LPUART7_ClockRoot_MuxSysPll2Pfd3 LPUART7 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART8_ClockRoot_MuxOscRc48MDiv2 LPUART8 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART8_ClockRoot_MuxOsc24MOut LPUART8 mux from MuxOsc24MOut.

kCLOCK_LPUART8_ClockRoot_MuxOscRc400M LPUART8 mux from MuxOscRc400M.

kCLOCK_LPUART8_ClockRoot_MuxOscRc16M LPUART8 mux from MuxOscRc16M.

kCLOCK_LPUART8_ClockRoot_MuxSysPll3Div2 LPUART8 mux from MuxSysPll3Div2.

kCLOCK_LPUART8_ClockRoot_MuxSysPll1Div5 LPUART8 mux from MuxSysPll1Div5.

kCLOCK_LPUART8_ClockRoot_MuxSysPll2Out LPUART8 mux from MuxSysPll2Out.

kCLOCK_LPUART8_ClockRoot_MuxSysPll2Pfd3 LPUART8 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART9_ClockRoot_MuxOscRc48MDiv2 LPUART9 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART9_ClockRoot_MuxOsc24MOut LPUART9 mux from MuxOsc24MOut.

kCLOCK_LPUART9_ClockRoot_MuxOscRc400M LPUART9 mux from MuxOscRc400M.

kCLOCK_LPUART9_ClockRoot_MuxOscRc16M LPUART9 mux from MuxOscRc16M.

kCLOCK_LPUART9_ClockRoot_MuxSysPll3Div2 LPUART9 mux from MuxSysPll3Div2.

kCLOCK_LPUART9_ClockRoot_MuxSysPll1Div5 LPUART9 mux from MuxSysPll1Div5.

kCLOCK_LPUART9_ClockRoot_MuxSysPll2Out LPUART9 mux from MuxSysPll2Out.

kCLOCK_LPUART9_ClockRoot_MuxSysPll2Pfd3 LPUART9 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART10_ClockRoot_MuxOscRc48MDiv2 LPUART10 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART10_ClockRoot_MuxOsc24MOut LPUART10 mux from MuxOsc24MOut.

kCLOCK_LPUART10_ClockRoot_MuxOscRc400M LPUART10 mux from MuxOscRc400M.

kCLOCK_LPUART10_ClockRoot_MuxOscRc16M LPUART10 mux from MuxOscRc16M.

kCLOCK_LPUART10_ClockRoot_MuxSysPll3Div2 LPUART10 mux from MuxSysPll3Div2.

kCLOCK_LPUART10_ClockRoot_MuxSysPll1Div5 LPUART10 mux from MuxSysPll1Div5.

kCLOCK_LPUART10_ClockRoot_MuxSysPll2Out LPUART10 mux from MuxSysPll2Out.

kCLOCK_LPUART10_ClockRoot_MuxSysPll2Pfd3 LPUART10 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART11_ClockRoot_MuxOscRc48MDiv2 LPUART11 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART11_ClockRoot_MuxOsc24MOut LPUART11 mux from MuxOsc24MOut.

kCLOCK_LPUART11_ClockRoot_MuxOscRc400M LPUART11 mux from MuxOscRc400M.

kCLOCK_LPUART11_ClockRoot_MuxOscRc16M LPUART11 mux from MuxOscRc16M.

kCLOCK_LPUART11_ClockRoot_MuxSysPll3Pfd3 LPUART11 mux from MuxSysPll3Pfd3.

kCLOCK_LPUART11_ClockRoot_MuxSysPll3Out LPUART11 mux from MuxSysPll3Out.

kCLOCK_LPUART11_ClockRoot_MuxSysPll2Pfd3 LPUART11 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART11_ClockRoot_MuxSysPll1Div5 LPUART11 mux from MuxSysPll1Div5.

kCLOCK_LPUART12_ClockRoot_MuxOscRc48MDiv2 LPUART12 mux from MuxOscRc48M-Div2.

kCLOCK_LPUART12_ClockRoot_MuxOsc24MOut LPUART12 mux from MuxOsc24MOut.

kCLOCK_LPUART12_ClockRoot_MuxOscRc400M LPUART12 mux from MuxOscRc400M.

kCLOCK_LPUART12_ClockRoot_MuxOscRc16M LPUART12 mux from MuxOscRc16M.

kCLOCK_LPUART12_ClockRoot_MuxSysPll3Pfd3 LPUART12 mux from MuxSysPll3Pfd3.

kCLOCK_LPUART12_ClockRoot_MuxSysPll3Out LPUART12 mux from MuxSysPll3Out.

kCLOCK_LPUART12_ClockRoot_MuxSysPll2Pfd3 LPUART12 mux from MuxSysPll2Pfd3.

kCLOCK_LPUART12_ClockRoot_MuxSysPll1Div5 LPUART12 mux from MuxSysPll1Div5.

kCLOCK_LPI2C1_ClockRoot_MuxOscRc48MDiv2 LPI2C1 mux from MuxOscRc48MDiv2.

kCLOCK_LPI2C1_ClockRoot_MuxOsc24MOut LPI2C1 mux from MuxOsc24MOut.

kCLOCK_LPI2C1_ClockRoot_MuxOscRc400M LPI2C1 mux from MuxOscRc400M.

kCLOCK_LPI2C1_ClockRoot_MuxOscRc16M LPI2C1 mux from MuxOscRc16M.

kCLOCK_LPI2C1_ClockRoot_MuxSysPll3Div2 LPI2C1 mux from MuxSysPll3Div2.

kCLOCK_LPI2C1_ClockRoot_MuxSysPll1Div5 LPI2C1 mux from MuxSysPll1Div5.

kCLOCK_LPI2C1_ClockRoot_MuxSysPll2Out LPI2C1 mux from MuxSysPll2Out.

kCLOCK_LPI2C1_ClockRoot_MuxSysPll2Pfd3 LPI2C1 mux from MuxSysPll2Pfd3.

kCLOCK_LPI2C2_ClockRoot_MuxOscRc48MDiv2 LPI2C2 mux from MuxOscRc48MDiv2.

kCLOCK_LPI2C2_ClockRoot_MuxOsc24MOut LPI2C2 mux from MuxOsc24MOut.

kCLOCK_LPI2C2_ClockRoot_MuxOscRc400M LPI2C2 mux from MuxOscRc400M.

kCLOCK_LPI2C2_ClockRoot_MuxOscRc16M LPI2C2 mux from MuxOscRc16M.

kCLOCK_LPI2C2_ClockRoot_MuxSysPll3Div2 LPI2C2 mux from MuxSysPll3Div2.

kCLOCK_LPI2C2_ClockRoot_MuxSysPll1Div5 LPI2C2 mux from MuxSysPll1Div5.

kCLOCK_LPI2C2_ClockRoot_MuxSysPll2Out LPI2C2 mux from MuxSysPll2Out.

kCLOCK_LPI2C2_ClockRoot_MuxSysPll2Pfd3 LPI2C2 mux from MuxSysPll2Pfd3.

kCLOCK_LPI2C3_ClockRoot_MuxOscRc48MDiv2 LPI2C3 mux from MuxOscRc48MDiv2.

kCLOCK_LPI2C3_ClockRoot_MuxOsc24MOut LPI2C3 mux from MuxOsc24MOut.

kCLOCK_LPI2C3_ClockRoot_MuxOscRc400M LPI2C3 mux from MuxOscRc400M.

kCLOCK_LPI2C3_ClockRoot_MuxOscRc16M LPI2C3 mux from MuxOscRc16M.

kCLOCK_LPI2C3_ClockRoot_MuxSysPll3Div2 LPI2C3 mux from MuxSysPll3Div2.

kCLOCK_LPI2C3_ClockRoot_MuxSysPll1Div5 LPI2C3 mux from MuxSysPll1Div5.

kCLOCK_LPI2C3_ClockRoot_MuxSysPll2Out LPI2C3 mux from MuxSysPll2Out.

kCLOCK_LPI2C3_ClockRoot_MuxSysPll2Pfd3 LPI2C3 mux from MuxSysPll2Pfd3.

kCLOCK_LPI2C4_ClockRoot_MuxOscRc48MDiv2 LPI2C4 mux from MuxOscRc48MDiv2.

kCLOCK_LPI2C4_ClockRoot_MuxOsc24MOut LPI2C4 mux from MuxOsc24MOut.

kCLOCK_LPI2C4_ClockRoot_MuxOscRc400M LPI2C4 mux from MuxOscRc400M.

kCLOCK_LPI2C4_ClockRoot_MuxOscRc16M LPI2C4 mux from MuxOscRc16M.

kCLOCK_LPI2C4_ClockRoot_MuxSysPll3Div2 LPI2C4 mux from MuxSysPll3Div2.

kCLOCK_LPI2C4_ClockRoot_MuxSysPll1Div5 LPI2C4 mux from MuxSysPll1Div5.

kCLOCK_LPI2C4_ClockRoot_MuxSysPll2Out LPI2C4 mux from MuxSysPll2Out.

kCLOCK_LPI2C4_ClockRoot_MuxSysPll2Pfd3 LPI2C4 mux from MuxSysPll2Pfd3.

kCLOCK_LPI2C5_ClockRoot_MuxOscRc48MDiv2 LPI2C5 mux from MuxOscRc48MDiv2.

kCLOCK_LPI2C5_ClockRoot_MuxOsc24MOut LPI2C5 mux from MuxOsc24MOut.

kCLOCK_LPI2C5_ClockRoot_MuxOscRc400M LPI2C5 mux from MuxOscRc400M.

<i>kCLOCK_LPI2C5_ClockRoot_MuxOscRc16M</i>	LPI2C5 mux from MuxOscRc16M.
<i>kCLOCK_LPI2C5_ClockRoot_MuxSysPll3Pfd3</i>	LPI2C5 mux from MuxSysPll3Pfd3.
<i>kCLOCK_LPI2C5_ClockRoot_MuxSysPll3Out</i>	LPI2C5 mux from MuxSysPll3Out.
<i>kCLOCK_LPI2C5_ClockRoot_MuxSysPll2Pfd3</i>	LPI2C5 mux from MuxSysPll2Pfd3.
<i>kCLOCK_LPI2C5_ClockRoot_MuxSysPll1Div5</i>	LPI2C5 mux from MuxSysPll1Div5.
<i>kCLOCK_LPI2C6_ClockRoot_MuxOscRc48MDiv2</i>	LPI2C6 mux from MuxOscRc48MDiv2.
<i>kCLOCK_LPI2C6_ClockRoot_MuxOsc24MOut</i>	LPI2C6 mux from MuxOsc24MOut.
<i>kCLOCK_LPI2C6_ClockRoot_MuxOscRc400M</i>	LPI2C6 mux from MuxOscRc400M.
<i>kCLOCK_LPI2C6_ClockRoot_MuxOscRc16M</i>	LPI2C6 mux from MuxOscRc16M.
<i>kCLOCK_LPI2C6_ClockRoot_MuxSysPll3Pfd3</i>	LPI2C6 mux from MuxSysPll3Pfd3.
<i>kCLOCK_LPI2C6_ClockRoot_MuxSysPll3Out</i>	LPI2C6 mux from MuxSysPll3Out.
<i>kCLOCK_LPI2C6_ClockRoot_MuxSysPll2Pfd3</i>	LPI2C6 mux from MuxSysPll2Pfd3.
<i>kCLOCK_LPI2C6_ClockRoot_MuxSysPll1Div5</i>	LPI2C6 mux from MuxSysPll1Div5.
<i>kCLOCK_LPSP11_ClockRoot_MuxOscRc48MDiv2</i>	LPSP11 mux from MuxOscRc48MDiv2.
<i>kCLOCK_LPSP11_ClockRoot_MuxOsc24MOut</i>	LPSP11 mux from MuxOsc24MOut.
<i>kCLOCK_LPSP11_ClockRoot_MuxOscRc400M</i>	LPSP11 mux from MuxOscRc400M.
<i>kCLOCK_LPSP11_ClockRoot_MuxOscRc16M</i>	LPSP11 mux from MuxOscRc16M.
<i>kCLOCK_LPSP11_ClockRoot_MuxSysPll3Pfd2</i>	LPSP11 mux from MuxSysPll3Pfd2.
<i>kCLOCK_LPSP11_ClockRoot_MuxSysPll1Div5</i>	LPSP11 mux from MuxSysPll1Div5.
<i>kCLOCK_LPSP11_ClockRoot_MuxSysPll2Out</i>	LPSP11 mux from MuxSysPll2Out.
<i>kCLOCK_LPSP11_ClockRoot_MuxSysPll2Pfd3</i>	LPSP11 mux from MuxSysPll2Pfd3.
<i>kCLOCK_LPSP12_ClockRoot_MuxOscRc48MDiv2</i>	LPSP12 mux from MuxOscRc48MDiv2.
<i>kCLOCK_LPSP12_ClockRoot_MuxOsc24MOut</i>	LPSP12 mux from MuxOsc24MOut.
<i>kCLOCK_LPSP12_ClockRoot_MuxOscRc400M</i>	LPSP12 mux from MuxOscRc400M.
<i>kCLOCK_LPSP12_ClockRoot_MuxOscRc16M</i>	LPSP12 mux from MuxOscRc16M.
<i>kCLOCK_LPSP12_ClockRoot_MuxSysPll3Pfd2</i>	LPSP12 mux from MuxSysPll3Pfd2.
<i>kCLOCK_LPSP12_ClockRoot_MuxSysPll1Div5</i>	LPSP12 mux from MuxSysPll1Div5.
<i>kCLOCK_LPSP12_ClockRoot_MuxSysPll2Out</i>	LPSP12 mux from MuxSysPll2Out.
<i>kCLOCK_LPSP12_ClockRoot_MuxSysPll2Pfd3</i>	LPSP12 mux from MuxSysPll2Pfd3.
<i>kCLOCK_LPSP13_ClockRoot_MuxOscRc48MDiv2</i>	LPSP13 mux from MuxOscRc48MDiv2.
<i>kCLOCK_LPSP13_ClockRoot_MuxOsc24MOut</i>	LPSP13 mux from MuxOsc24MOut.
<i>kCLOCK_LPSP13_ClockRoot_MuxOscRc400M</i>	LPSP13 mux from MuxOscRc400M.
<i>kCLOCK_LPSP13_ClockRoot_MuxOscRc16M</i>	LPSP13 mux from MuxOscRc16M.
<i>kCLOCK_LPSP13_ClockRoot_MuxSysPll3Pfd2</i>	LPSP13 mux from MuxSysPll3Pfd2.
<i>kCLOCK_LPSP13_ClockRoot_MuxSysPll1Div5</i>	LPSP13 mux from MuxSysPll1Div5.
<i>kCLOCK_LPSP13_ClockRoot_MuxSysPll2Out</i>	LPSP13 mux from MuxSysPll2Out.
<i>kCLOCK_LPSP13_ClockRoot_MuxSysPll2Pfd3</i>	LPSP13 mux from MuxSysPll2Pfd3.
<i>kCLOCK_LPSP14_ClockRoot_MuxOscRc48MDiv2</i>	LPSP14 mux from MuxOscRc48MDiv2.
<i>kCLOCK_LPSP14_ClockRoot_MuxOsc24MOut</i>	LPSP14 mux from MuxOsc24MOut.
<i>kCLOCK_LPSP14_ClockRoot_MuxOscRc400M</i>	LPSP14 mux from MuxOscRc400M.
<i>kCLOCK_LPSP14_ClockRoot_MuxOscRc16M</i>	LPSP14 mux from MuxOscRc16M.
<i>kCLOCK_LPSP14_ClockRoot_MuxSysPll3Pfd2</i>	LPSP14 mux from MuxSysPll3Pfd2.
<i>kCLOCK_LPSP14_ClockRoot_MuxSysPll1Div5</i>	LPSP14 mux from MuxSysPll1Div5.
<i>kCLOCK_LPSP14_ClockRoot_MuxSysPll2Out</i>	LPSP14 mux from MuxSysPll2Out.
<i>kCLOCK_LPSP14_ClockRoot_MuxSysPll2Pfd3</i>	LPSP14 mux from MuxSysPll2Pfd3.

kCLOCK_LPSPi5_ClockRoot_MuxOscRc48MDiv2 LPSPi5 mux from MuxOscRc48MDiv2.
kCLOCK_LPSPi5_ClockRoot_MuxOsc24MOut LPSPi5 mux from MuxOsc24MOut.
kCLOCK_LPSPi5_ClockRoot_MuxOscRc400M LPSPi5 mux from MuxOscRc400M.
kCLOCK_LPSPi5_ClockRoot_MuxOscRc16M LPSPi5 mux from MuxOscRc16M.
kCLOCK_LPSPi5_ClockRoot_MuxSysPll3Pfd3 LPSPi5 mux from MuxSysPll3Pfd3.
kCLOCK_LPSPi5_ClockRoot_MuxSysPll3Out LPSPi5 mux from MuxSysPll3Out.
kCLOCK_LPSPi5_ClockRoot_MuxSysPll3Pfd2 LPSPi5 mux from MuxSysPll3Pfd2.
kCLOCK_LPSPi5_ClockRoot_MuxSysPll1Div5 LPSPi5 mux from MuxSysPll1Div5.
kCLOCK_LPSPi6_ClockRoot_MuxOscRc48MDiv2 LPSPi6 mux from MuxOscRc48MDiv2.
kCLOCK_LPSPi6_ClockRoot_MuxOsc24MOut LPSPi6 mux from MuxOsc24MOut.
kCLOCK_LPSPi6_ClockRoot_MuxOscRc400M LPSPi6 mux from MuxOscRc400M.
kCLOCK_LPSPi6_ClockRoot_MuxOscRc16M LPSPi6 mux from MuxOscRc16M.
kCLOCK_LPSPi6_ClockRoot_MuxSysPll3Pfd3 LPSPi6 mux from MuxSysPll3Pfd3.
kCLOCK_LPSPi6_ClockRoot_MuxSysPll3Out LPSPi6 mux from MuxSysPll3Out.
kCLOCK_LPSPi6_ClockRoot_MuxSysPll3Pfd2 LPSPi6 mux from MuxSysPll3Pfd2.
kCLOCK_LPSPi6_ClockRoot_MuxSysPll1Div5 LPSPi6 mux from MuxSysPll1Div5.
kCLOCK_EMV1_ClockRoot_MuxOscRc48MDiv2 EMV1 mux from MuxOscRc48MDiv2.
kCLOCK_EMV1_ClockRoot_MuxOsc24MOut EMV1 mux from MuxOsc24MOut.
kCLOCK_EMV1_ClockRoot_MuxOscRc400M EMV1 mux from MuxOscRc400M.
kCLOCK_EMV1_ClockRoot_MuxOscRc16M EMV1 mux from MuxOscRc16M.
kCLOCK_EMV1_ClockRoot_MuxSysPll3Div2 EMV1 mux from MuxSysPll3Div2.
kCLOCK_EMV1_ClockRoot_MuxSysPll1Div5 EMV1 mux from MuxSysPll1Div5.
kCLOCK_EMV1_ClockRoot_MuxSysPll2Out EMV1 mux from MuxSysPll2Out.
kCLOCK_EMV1_ClockRoot_MuxSysPll2Pfd3 EMV1 mux from MuxSysPll2Pfd3.
kCLOCK_EMV2_ClockRoot_MuxOscRc48MDiv2 EMV2 mux from MuxOscRc48MDiv2.
kCLOCK_EMV2_ClockRoot_MuxOsc24MOut EMV2 mux from MuxOsc24MOut.
kCLOCK_EMV2_ClockRoot_MuxOscRc400M EMV2 mux from MuxOscRc400M.
kCLOCK_EMV2_ClockRoot_MuxOscRc16M EMV2 mux from MuxOscRc16M.
kCLOCK_EMV2_ClockRoot_MuxSysPll3Div2 EMV2 mux from MuxSysPll3Div2.
kCLOCK_EMV2_ClockRoot_MuxSysPll1Div5 EMV2 mux from MuxSysPll1Div5.
kCLOCK_EMV2_ClockRoot_MuxSysPll2Out EMV2 mux from MuxSysPll2Out.
kCLOCK_EMV2_ClockRoot_MuxSysPll2Pfd3 EMV2 mux from MuxSysPll2Pfd3.
kCLOCK_ENET1_ClockRoot_MuxOscRc48MDiv2 ENET1 mux from MuxOscRc48MDiv2.
kCLOCK_ENET1_ClockRoot_MuxOsc24MOut ENET1 mux from MuxOsc24MOut.
kCLOCK_ENET1_ClockRoot_MuxOscRc400M ENET1 mux from MuxOscRc400M.
kCLOCK_ENET1_ClockRoot_MuxOscRc16M ENET1 mux from MuxOscRc16M.
kCLOCK_ENET1_ClockRoot_MuxSysPll1Div2 ENET1 mux from MuxSysPll1Div2.
kCLOCK_ENET1_ClockRoot_MuxAudioPllOut ENET1 mux from MuxAudioPllOut.
kCLOCK_ENET1_ClockRoot_MuxSysPll1Div5 ENET1 mux from MuxSysPll1Div5.
kCLOCK_ENET1_ClockRoot_MuxSysPll2Pfd1 ENET1 mux from MuxSysPll2Pfd1.
kCLOCK_ENET2_ClockRoot_MuxOscRc48MDiv2 ENET2 mux from MuxOscRc48MDiv2.
kCLOCK_ENET2_ClockRoot_MuxOsc24MOut ENET2 mux from MuxOsc24MOut.
kCLOCK_ENET2_ClockRoot_MuxOscRc400M ENET2 mux from MuxOscRc400M.
kCLOCK_ENET2_ClockRoot_MuxOscRc16M ENET2 mux from MuxOscRc16M.
kCLOCK_ENET2_ClockRoot_MuxSysPll1Div2 ENET2 mux from MuxSysPll1Div2.

kCLOCK_ENET2_ClockRoot_MuxAudioPllOut ENET2 mux from MuxAudioPllOut.

kCLOCK_ENET2_ClockRoot_MuxSysPll1Div5 ENET2 mux from MuxSysPll1Div5.

kCLOCK_ENET2_ClockRoot_MuxSysPll2Pfd1 ENET2 mux from MuxSysPll2Pfd1.

kCLOCK_ENET_QOS_ClockRoot_MuxOscRc48MDiv2 ENET_QOS mux from MuxOscRc48M-Div2.

kCLOCK_ENET_QOS_ClockRoot_MuxOsc24MOut ENET_QOS mux from MuxOsc24MOut.

kCLOCK_ENET_QOS_ClockRoot_MuxOscRc400M ENET_QOS mux from MuxOscRc400M.

kCLOCK_ENET_QOS_ClockRoot_MuxOscRc16M ENET_QOS mux from MuxOscRc16M.

kCLOCK_ENET_QOS_ClockRoot_MuxSysPll1Div2 ENET_QOS mux from MuxSysPll1Div2.

kCLOCK_ENET_QOS_ClockRoot_MuxAudioPllOut ENET_QOS mux from MuxAudioPllOut.

kCLOCK_ENET_QOS_ClockRoot_MuxSysPll1Div5 ENET_QOS mux from MuxSysPll1Div5.

kCLOCK_ENET_QOS_ClockRoot_MuxSysPll2Pfd1 ENET_QOS mux from MuxSysPll2Pfd1.

kCLOCK_ENET_25M_ClockRoot_MuxOscRc48MDiv2 ENET_25M mux from MuxOscRc48M-Div2.

kCLOCK_ENET_25M_ClockRoot_MuxOsc24MOut ENET_25M mux from MuxOsc24MOut.

kCLOCK_ENET_25M_ClockRoot_MuxOscRc400M ENET_25M mux from MuxOscRc400M.

kCLOCK_ENET_25M_ClockRoot_MuxOscRc16M ENET_25M mux from MuxOscRc16M.

kCLOCK_ENET_25M_ClockRoot_MuxSysPll1Div2 ENET_25M mux from MuxSysPll1Div2.

kCLOCK_ENET_25M_ClockRoot_MuxAudioPllOut ENET_25M mux from MuxAudioPllOut.

kCLOCK_ENET_25M_ClockRoot_MuxSysPll1Div5 ENET_25M mux from MuxSysPll1Div5.

kCLOCK_ENET_25M_ClockRoot_MuxSysPll2Pfd1 ENET_25M mux from MuxSysPll2Pfd1.

kCLOCK_ENET_TIMER1_ClockRoot_MuxOscRc48MDiv2 ENET_TIMER1 mux from Mux-OscRc48MDiv2.

kCLOCK_ENET_TIMER1_ClockRoot_MuxOsc24MOut ENET_TIMER1 mux from MuxOsc24-MOut.

kCLOCK_ENET_TIMER1_ClockRoot_MuxOscRc400M ENET_TIMER1 mux from MuxOsc-Rc400M.

kCLOCK_ENET_TIMER1_ClockRoot_MuxOscRc16M ENET_TIMER1 mux from MuxOsc-Rc16M.

kCLOCK_ENET_TIMER1_ClockRoot_MuxSysPll1Div2 ENET_TIMER1 mux from MuxSys-Pll1Div2.

kCLOCK_ENET_TIMER1_ClockRoot_MuxAudioPllOut ENET_TIMER1 mux from MuxAudio-PllOut.

kCLOCK_ENET_TIMER1_ClockRoot_MuxSysPll1Div5 ENET_TIMER1 mux from MuxSys-Pll1Div5.

kCLOCK_ENET_TIMER1_ClockRoot_MuxSysPll2Pfd1 ENET_TIMER1 mux from MuxSys-Pll2Pfd1.

kCLOCK_ENET_TIMER2_ClockRoot_MuxOscRc48MDiv2 ENET_TIMER2 mux from Mux-OscRc48MDiv2.

kCLOCK_ENET_TIMER2_ClockRoot_MuxOsc24MOut ENET_TIMER2 mux from MuxOsc24-MOut.

kCLOCK_ENET_TIMER2_ClockRoot_MuxOscRc400M ENET_TIMER2 mux from MuxOsc-Rc400M.

kCLOCK_ENET_TIMER2_ClockRoot_MuxOscRc16M ENET_TIMER2 mux from MuxOsc-Rc16M.

kCLOCK_ENET_TIMER2_ClockRoot_MuxSysPll1Div2 ENET_TIMER2 mux from MuxSysPll1Div2.

kCLOCK_ENET_TIMER2_ClockRoot_MuxAudioPllOut ENET_TIMER2 mux from MuxAudioPllOut.

kCLOCK_ENET_TIMER2_ClockRoot_MuxSysPll1Div5 ENET_TIMER2 mux from MuxSysPll1Div5.

kCLOCK_ENET_TIMER2_ClockRoot_MuxSysPll2Pfd1 ENET_TIMER2 mux from MuxSysPll2Pfd1.

kCLOCK_ENET_TIMER3_ClockRoot_MuxOscRc48MDiv2 ENET_TIMER3 mux from MuxOscRc48MDiv2.

kCLOCK_ENET_TIMER3_ClockRoot_MuxOsc24MOut ENET_TIMER3 mux from MuxOsc24MOut.

kCLOCK_ENET_TIMER3_ClockRoot_MuxOscRc400M ENET_TIMER3 mux from MuxOscRc400M.

kCLOCK_ENET_TIMER3_ClockRoot_MuxOscRc16M ENET_TIMER3 mux from MuxOscRc16M.

kCLOCK_ENET_TIMER3_ClockRoot_MuxSysPll1Div2 ENET_TIMER3 mux from MuxSysPll1Div2.

kCLOCK_ENET_TIMER3_ClockRoot_MuxAudioPllOut ENET_TIMER3 mux from MuxAudioPllOut.

kCLOCK_ENET_TIMER3_ClockRoot_MuxSysPll1Div5 ENET_TIMER3 mux from MuxSysPll1Div5.

kCLOCK_ENET_TIMER3_ClockRoot_MuxSysPll2Pfd1 ENET_TIMER3 mux from MuxSysPll2Pfd1.

kCLOCK_USDHC1_ClockRoot_MuxOscRc48MDiv2 USDHC1 mux from MuxOscRc48MDiv2.

kCLOCK_USDHC1_ClockRoot_MuxOsc24MOut USDHC1 mux from MuxOsc24MOut.

kCLOCK_USDHC1_ClockRoot_MuxOscRc400M USDHC1 mux from MuxOscRc400M.

kCLOCK_USDHC1_ClockRoot_MuxOscRc16M USDHC1 mux from MuxOscRc16M.

kCLOCK_USDHC1_ClockRoot_MuxSysPll2Pfd2 USDHC1 mux from MuxSysPll2Pfd2.

kCLOCK_USDHC1_ClockRoot_MuxSysPll2Pfd0 USDHC1 mux from MuxSysPll2Pfd0.

kCLOCK_USDHC1_ClockRoot_MuxSysPll1Div5 USDHC1 mux from MuxSysPll1Div5.

kCLOCK_USDHC1_ClockRoot_MuxArmPllOut USDHC1 mux from MuxArmPllOut.

kCLOCK_USDHC2_ClockRoot_MuxOscRc48MDiv2 USDHC2 mux from MuxOscRc48MDiv2.

kCLOCK_USDHC2_ClockRoot_MuxOsc24MOut USDHC2 mux from MuxOsc24MOut.

kCLOCK_USDHC2_ClockRoot_MuxOscRc400M USDHC2 mux from MuxOscRc400M.

kCLOCK_USDHC2_ClockRoot_MuxOscRc16M USDHC2 mux from MuxOscRc16M.

kCLOCK_USDHC2_ClockRoot_MuxSysPll2Pfd2 USDHC2 mux from MuxSysPll2Pfd2.

kCLOCK_USDHC2_ClockRoot_MuxSysPll2Pfd0 USDHC2 mux from MuxSysPll2Pfd0.

kCLOCK_USDHC2_ClockRoot_MuxSysPll1Div5 USDHC2 mux from MuxSysPll1Div5.

kCLOCK_USDHC2_ClockRoot_MuxArmPllOut USDHC2 mux from MuxArmPllOut.

kCLOCK_ASRC_ClockRoot_MuxOscRc48MDiv2 ASRC mux from MuxOscRc48MDiv2.

kCLOCK_ASRC_ClockRoot_MuxOsc24MOut ASRC mux from MuxOsc24MOut.

kCLOCK_ASRC_ClockRoot_MuxOscRc400M ASRC mux from MuxOscRc400M.

kCLOCK_ASRC_ClockRoot_MuxOscRc16M ASRC mux from MuxOscRc16M.

kCLOCK_ASRC_ClockRoot_MuxSysPll1Div5 ASRC mux from MuxSysPll1Div5.

kCLOCK_ASRC_ClockRoot_MuxSysPll3Div2 ASRC mux from MuxSysPll3Div2.
kCLOCK_ASRC_ClockRoot_MuxAudioPllOut ASRC mux from MuxAudioPllOut.
kCLOCK_ASRC_ClockRoot_MuxSysPll2Pfd3 ASRC mux from MuxSysPll2Pfd3.
kCLOCK_MQS_ClockRoot_MuxOscRc48MDiv2 MQS mux from MuxOscRc48MDiv2.
kCLOCK_MQS_ClockRoot_MuxOsc24MOut MQS mux from MuxOsc24MOut.
kCLOCK_MQS_ClockRoot_MuxOscRc400M MQS mux from MuxOscRc400M.
kCLOCK_MQS_ClockRoot_MuxOscRc16M MQS mux from MuxOscRc16M.
kCLOCK_MQS_ClockRoot_MuxSysPll1Div5 MQS mux from MuxSysPll1Div5.
kCLOCK_MQS_ClockRoot_MuxSysPll3Div2 MQS mux from MuxSysPll3Div2.
kCLOCK_MQS_ClockRoot_MuxAudioPllOut MQS mux from MuxAudioPllOut.
kCLOCK_MQS_ClockRoot_MuxSysPll2Pfd3 MQS mux from MuxSysPll2Pfd3.
kCLOCK_MIC_ClockRoot_MuxOscRc48MDiv2 MIC mux from MuxOscRc48MDiv2.
kCLOCK_MIC_ClockRoot_MuxOsc24MOut MIC mux from MuxOsc24MOut.
kCLOCK_MIC_ClockRoot_MuxOscRc400M MIC mux from MuxOscRc400M.
kCLOCK_MIC_ClockRoot_MuxOscRc16M MIC mux from MuxOscRc16M.
kCLOCK_MIC_ClockRoot_MuxSysPll3Pfd3 MIC mux from MuxSysPll3Pfd3.
kCLOCK_MIC_ClockRoot_MuxSysPll3Out MIC mux from MuxSysPll3Out.
kCLOCK_MIC_ClockRoot_MuxAudioPllOut MIC mux from MuxAudioPllOut.
kCLOCK_MIC_ClockRoot_MuxSysPll1Div5 MIC mux from MuxSysPll1Div5.
kCLOCK_SPDIF_ClockRoot_MuxOscRc48MDiv2 SPDIF mux from MuxOscRc48MDiv2.
kCLOCK_SPDIF_ClockRoot_MuxOsc24MOut SPDIF mux from MuxOsc24MOut.
kCLOCK_SPDIF_ClockRoot_MuxOscRc400M SPDIF mux from MuxOscRc400M.
kCLOCK_SPDIF_ClockRoot_MuxOscRc16M SPDIF mux from MuxOscRc16M.
kCLOCK_SPDIF_ClockRoot_MuxAudioPllOut SPDIF mux from MuxAudioPllOut.
kCLOCK_SPDIF_ClockRoot_MuxSysPll3Out SPDIF mux from MuxSysPll3Out.
kCLOCK_SPDIF_ClockRoot_MuxSysPll3Pfd2 SPDIF mux from MuxSysPll3Pfd2.
kCLOCK_SPDIF_ClockRoot_MuxSysPll2Pfd3 SPDIF mux from MuxSysPll2Pfd3.
kCLOCK_SAI1_ClockRoot_MuxOscRc48MDiv2 SAI1 mux from MuxOscRc48MDiv2.
kCLOCK_SAI1_ClockRoot_MuxOsc24MOut SAI1 mux from MuxOsc24MOut.
kCLOCK_SAI1_ClockRoot_MuxOscRc400M SAI1 mux from MuxOscRc400M.
kCLOCK_SAI1_ClockRoot_MuxOscRc16M SAI1 mux from MuxOscRc16M.
kCLOCK_SAI1_ClockRoot_MuxAudioPllOut SAI1 mux from MuxAudioPllOut.
kCLOCK_SAI1_ClockRoot_MuxSysPll3Pfd2 SAI1 mux from MuxSysPll3Pfd2.
kCLOCK_SAI1_ClockRoot_MuxSysPll1Div5 SAI1 mux from MuxSysPll1Div5.
kCLOCK_SAI1_ClockRoot_MuxSysPll2Pfd3 SAI1 mux from MuxSysPll2Pfd3.
kCLOCK_SAI2_ClockRoot_MuxOscRc48MDiv2 SAI2 mux from MuxOscRc48MDiv2.
kCLOCK_SAI2_ClockRoot_MuxOsc24MOut SAI2 mux from MuxOsc24MOut.
kCLOCK_SAI2_ClockRoot_MuxOscRc400M SAI2 mux from MuxOscRc400M.
kCLOCK_SAI2_ClockRoot_MuxOscRc16M SAI2 mux from MuxOscRc16M.
kCLOCK_SAI2_ClockRoot_MuxAudioPllOut SAI2 mux from MuxAudioPllOut.
kCLOCK_SAI2_ClockRoot_MuxSysPll3Pfd2 SAI2 mux from MuxSysPll3Pfd2.
kCLOCK_SAI2_ClockRoot_MuxSysPll1Div5 SAI2 mux from MuxSysPll1Div5.
kCLOCK_SAI2_ClockRoot_MuxSysPll2Pfd3 SAI2 mux from MuxSysPll2Pfd3.
kCLOCK_SAI3_ClockRoot_MuxOscRc48MDiv2 SAI3 mux from MuxOscRc48MDiv2.
kCLOCK_SAI3_ClockRoot_MuxOsc24MOut SAI3 mux from MuxOsc24MOut.

kCLOCK_SAI3_ClockRoot_MuxOscRc400M SAI3 mux from MuxOscRc400M.
kCLOCK_SAI3_ClockRoot_MuxOscRc16M SAI3 mux from MuxOscRc16M.
kCLOCK_SAI3_ClockRoot_MuxAudioPllOut SAI3 mux from MuxAudioPllOut.
kCLOCK_SAI3_ClockRoot_MuxSysPll3Pfd2 SAI3 mux from MuxSysPll3Pfd2.
kCLOCK_SAI3_ClockRoot_MuxSysPll1Div5 SAI3 mux from MuxSysPll1Div5.
kCLOCK_SAI3_ClockRoot_MuxSysPll2Pfd3 SAI3 mux from MuxSysPll2Pfd3.
kCLOCK_SAI4_ClockRoot_MuxOscRc48MDiv2 SAI4 mux from MuxOscRc48MDiv2.
kCLOCK_SAI4_ClockRoot_MuxOsc24MOut SAI4 mux from MuxOsc24MOut.
kCLOCK_SAI4_ClockRoot_MuxOscRc400M SAI4 mux from MuxOscRc400M.
kCLOCK_SAI4_ClockRoot_MuxOscRc16M SAI4 mux from MuxOscRc16M.
kCLOCK_SAI4_ClockRoot_MuxSysPll3Pfd3 SAI4 mux from MuxSysPll3Pfd3.
kCLOCK_SAI4_ClockRoot_MuxSysPll3Out SAI4 mux from MuxSysPll3Out.
kCLOCK_SAI4_ClockRoot_MuxAudioPllOut SAI4 mux from MuxAudioPllOut.
kCLOCK_SAI4_ClockRoot_MuxSysPll1Div5 SAI4 mux from MuxSysPll1Div5.
kCLOCK_GC355_ClockRoot_MuxOscRc48MDiv2 GC355 mux from MuxOscRc48MDiv2.
kCLOCK_GC355_ClockRoot_MuxOsc24MOut GC355 mux from MuxOsc24MOut.
kCLOCK_GC355_ClockRoot_MuxOscRc400M GC355 mux from MuxOscRc400M.
kCLOCK_GC355_ClockRoot_MuxOscRc16M GC355 mux from MuxOscRc16M.
kCLOCK_GC355_ClockRoot_MuxSysPll2Out GC355 mux from MuxSysPll2Out.
kCLOCK_GC355_ClockRoot_MuxSysPll2Pfd1 GC355 mux from MuxSysPll2Pfd1.
kCLOCK_GC355_ClockRoot_MuxSysPll3Out GC355 mux from MuxSysPll3Out.
kCLOCK_GC355_ClockRoot_MuxVideoPllOut GC355 mux from MuxVideoPllOut.
kCLOCK_LCDIF_ClockRoot_MuxOscRc48MDiv2 LCDIF mux from MuxOscRc48MDiv2.
kCLOCK_LCDIF_ClockRoot_MuxOsc24MOut LCDIF mux from MuxOsc24MOut.
kCLOCK_LCDIF_ClockRoot_MuxOscRc400M LCDIF mux from MuxOscRc400M.
kCLOCK_LCDIF_ClockRoot_MuxOscRc16M LCDIF mux from MuxOscRc16M.
kCLOCK_LCDIF_ClockRoot_MuxSysPll2Out LCDIF mux from MuxSysPll2Out.
kCLOCK_LCDIF_ClockRoot_MuxSysPll2Pfd2 LCDIF mux from MuxSysPll2Pfd2.
kCLOCK_LCDIF_ClockRoot_MuxSysPll3Pfd0 LCDIF mux from MuxSysPll3Pfd0.
kCLOCK_LCDIF_ClockRoot_MuxVideoPllOut LCDIF mux from MuxVideoPllOut.
kCLOCK_LCDIFV2_ClockRoot_MuxOscRc48MDiv2 LCDIFV2 mux from MuxOscRc48MDiv2.

kCLOCK_LCDIFV2_ClockRoot_MuxOsc24MOut LCDIFV2 mux from MuxOsc24MOut.
kCLOCK_LCDIFV2_ClockRoot_MuxOscRc400M LCDIFV2 mux from MuxOscRc400M.
kCLOCK_LCDIFV2_ClockRoot_MuxOscRc16M LCDIFV2 mux from MuxOscRc16M.
kCLOCK_LCDIFV2_ClockRoot_MuxSysPll2Out LCDIFV2 mux from MuxSysPll2Out.
kCLOCK_LCDIFV2_ClockRoot_MuxSysPll2Pfd2 LCDIFV2 mux from MuxSysPll2Pfd2.
kCLOCK_LCDIFV2_ClockRoot_MuxSysPll3Pfd0 LCDIFV2 mux from MuxSysPll3Pfd0.
kCLOCK_LCDIFV2_ClockRoot_MuxVideoPllOut LCDIFV2 mux from MuxVideoPllOut.
kCLOCK_MIPI_REF_ClockRoot_MuxOscRc48MDiv2 MIPI_REF mux from MuxOscRc48M-
Div2.
kCLOCK_MIPI_REF_ClockRoot_MuxOsc24MOut MIPI_REF mux from MuxOsc24MOut.
kCLOCK_MIPI_REF_ClockRoot_MuxOscRc400M MIPI_REF mux from MuxOscRc400M.
kCLOCK_MIPI_REF_ClockRoot_MuxOscRc16M MIPI_REF mux from MuxOscRc16M.
kCLOCK_MIPI_REF_ClockRoot_MuxSysPll2Out MIPI_REF mux from MuxSysPll2Out.

kCLOCK_MIPI_REF_ClockRoot_MuxSysPll2Pfd0 MIPI_REF mux from MuxSysPll2Pfd0.

kCLOCK_MIPI_REF_ClockRoot_MuxSysPll3Pfd0 MIPI_REF mux from MuxSysPll3Pfd0.

kCLOCK_MIPI_REF_ClockRoot_MuxVideoPllOut MIPI_REF mux from MuxVideoPllOut.

kCLOCK_MIPI_ESC_ClockRoot_MuxOscRc48MDiv2 MIPI_ESC mux from MuxOscRc48M-Div2.

kCLOCK_MIPI_ESC_ClockRoot_MuxOsc24MOut MIPI_ESC mux from MuxOsc24MOut.

kCLOCK_MIPI_ESC_ClockRoot_MuxOscRc400M MIPI_ESC mux from MuxOscRc400M.

kCLOCK_MIPI_ESC_ClockRoot_MuxOscRc16M MIPI_ESC mux from MuxOscRc16M.

kCLOCK_MIPI_ESC_ClockRoot_MuxSysPll2Out MIPI_ESC mux from MuxSysPll2Out.

kCLOCK_MIPI_ESC_ClockRoot_MuxSysPll2Pfd0 MIPI_ESC mux from MuxSysPll2Pfd0.

kCLOCK_MIPI_ESC_ClockRoot_MuxSysPll3Pfd0 MIPI_ESC mux from MuxSysPll3Pfd0.

kCLOCK_MIPI_ESC_ClockRoot_MuxVideoPllOut MIPI_ESC mux from MuxVideoPllOut.

kCLOCK_CSI2_ClockRoot_MuxOscRc48MDiv2 CSI2 mux from MuxOscRc48MDiv2.

kCLOCK_CSI2_ClockRoot_MuxOsc24MOut CSI2 mux from MuxOsc24MOut.

kCLOCK_CSI2_ClockRoot_MuxOscRc400M CSI2 mux from MuxOscRc400M.

kCLOCK_CSI2_ClockRoot_MuxOscRc16M CSI2 mux from MuxOscRc16M.

kCLOCK_CSI2_ClockRoot_MuxSysPll2Pfd2 CSI2 mux from MuxSysPll2Pfd2.

kCLOCK_CSI2_ClockRoot_MuxSysPll3Out CSI2 mux from MuxSysPll3Out.

kCLOCK_CSI2_ClockRoot_MuxSysPll2Pfd0 CSI2 mux from MuxSysPll2Pfd0.

kCLOCK_CSI2_ClockRoot_MuxVideoPllOut CSI2 mux from MuxVideoPllOut.

kCLOCK_CSI2_ESC_ClockRoot_MuxOscRc48MDiv2 CSI2_ESC mux from MuxOscRc48M-Div2.

kCLOCK_CSI2_ESC_ClockRoot_MuxOsc24MOut CSI2_ESC mux from MuxOsc24MOut.

kCLOCK_CSI2_ESC_ClockRoot_MuxOscRc400M CSI2_ESC mux from MuxOscRc400M.

kCLOCK_CSI2_ESC_ClockRoot_MuxOscRc16M CSI2_ESC mux from MuxOscRc16M.

kCLOCK_CSI2_ESC_ClockRoot_MuxSysPll2Pfd2 CSI2_ESC mux from MuxSysPll2Pfd2.

kCLOCK_CSI2_ESC_ClockRoot_MuxSysPll3Out CSI2_ESC mux from MuxSysPll3Out.

kCLOCK_CSI2_ESC_ClockRoot_MuxSysPll2Pfd0 CSI2_ESC mux from MuxSysPll2Pfd0.

kCLOCK_CSI2_ESC_ClockRoot_MuxVideoPllOut CSI2_ESC mux from MuxVideoPllOut.

kCLOCK_CSI2_UI_ClockRoot_MuxOscRc48MDiv2 CSI2_UI mux from MuxOscRc48MDiv2.

kCLOCK_CSI2_UI_ClockRoot_MuxOsc24MOut CSI2_UI mux from MuxOsc24MOut.

kCLOCK_CSI2_UI_ClockRoot_MuxOscRc400M CSI2_UI mux from MuxOscRc400M.

kCLOCK_CSI2_UI_ClockRoot_MuxOscRc16M CSI2_UI mux from MuxOscRc16M.

kCLOCK_CSI2_UI_ClockRoot_MuxSysPll2Pfd2 CSI2_UI mux from MuxSysPll2Pfd2.

kCLOCK_CSI2_UI_ClockRoot_MuxSysPll3Out CSI2_UI mux from MuxSysPll3Out.

kCLOCK_CSI2_UI_ClockRoot_MuxSysPll2Pfd0 CSI2_UI mux from MuxSysPll2Pfd0.

kCLOCK_CSI2_UI_ClockRoot_MuxVideoPllOut CSI2_UI mux from MuxVideoPllOut.

kCLOCK_CSI_ClockRoot_MuxOscRc48MDiv2 CSI mux from MuxOscRc48MDiv2.

kCLOCK_CSI_ClockRoot_MuxOsc24MOut CSI mux from MuxOsc24MOut.

kCLOCK_CSI_ClockRoot_MuxOscRc400M CSI mux from MuxOscRc400M.

kCLOCK_CSI_ClockRoot_MuxOscRc16M CSI mux from MuxOscRc16M.

kCLOCK_CSI_ClockRoot_MuxSysPll2Pfd2 CSI mux from MuxSysPll2Pfd2.

kCLOCK_CSI_ClockRoot_MuxSysPll3Out CSI mux from MuxSysPll3Out.

kCLOCK_CSI_ClockRoot_MuxSysPll3Pfd1 CSI mux from MuxSysPll3Pfd1.

kCLOCK_CSI_ClockRoot_MuxVideoPllOut CSI mux from MuxVideoPllOut.

kCLOCK_CK01_ClockRoot_MuxOscRc48MDiv2 CKO1 mux from MuxOscRc48MDiv2.
kCLOCK_CK01_ClockRoot_MuxOsc24MOut CKO1 mux from MuxOsc24MOut.
kCLOCK_CK01_ClockRoot_MuxOscRc400M CKO1 mux from MuxOscRc400M.
kCLOCK_CK01_ClockRoot_MuxOscRc16M CKO1 mux from MuxOscRc16M.
kCLOCK_CK01_ClockRoot_MuxSysPll2Pfd2 CKO1 mux from MuxSysPll2Pfd2.
kCLOCK_CK01_ClockRoot_MuxSysPll2Out CKO1 mux from MuxSysPll2Out.
kCLOCK_CK01_ClockRoot_MuxSysPll3Pfd1 CKO1 mux from MuxSysPll3Pfd1.
kCLOCK_CK01_ClockRoot_MuxSysPll1Div5 CKO1 mux from MuxSysPll1Div5.
kCLOCK_CK02_ClockRoot_MuxOscRc48MDiv2 CKO2 mux from MuxOscRc48MDiv2.
kCLOCK_CK02_ClockRoot_MuxOsc24MOut CKO2 mux from MuxOsc24MOut.
kCLOCK_CK02_ClockRoot_MuxOscRc400M CKO2 mux from MuxOscRc400M.
kCLOCK_CK02_ClockRoot_MuxOscRc16M CKO2 mux from MuxOscRc16M.
kCLOCK_CK02_ClockRoot_MuxSysPll2Pfd3 CKO2 mux from MuxSysPll2Pfd3.
kCLOCK_CK02_ClockRoot_MuxOscRc48M CKO2 mux from MuxOscRc48M.
kCLOCK_CK02_ClockRoot_MuxSysPll3Pfd1 CKO2 mux from MuxSysPll3Pfd1.
kCLOCK_CK02_ClockRoot_MuxAudioPllOut CKO2 mux from MuxAudioPllOut.

5.4.5 enum clock_group_t

Enumerator

kCLOCK_Group_FlexRAM FlexRAM clock group.
kCLOCK_Group_MipiDsi Mipi Dsi clock group.
kCLOCK_Group_Last Last clock group.

5.4.6 enum clock_osc_t

Enumerator

kCLOCK_RcOsc On chip OSC.
kCLOCK_XtalOsc 24M Xtal OSC

5.4.7 enum clock_gate_value_t

Enumerator

kCLOCK_Off Clock is off.
kCLOCK_On Clock is on.

5.4.8 enum clock_mode_t

Enumerator

kCLOCK_ModeRun Remain in run mode.
kCLOCK_ModeWait Transfer to wait mode.
kCLOCK_ModeStop Transfer to stop mode.

5.4.9 enum clock_usb_src_t

Enumerator

kCLOCK_Usb480M Use 480M.
kCLOCK_UsbSrcUnused Used when the function does not care the clock source.

5.4.10 enum clock_usb_phy_src_t

Enumerator

kCLOCK_Usbphy480M Use 480M.

5.4.11 enum _clock_pll_clk_src

Enumerator

kCLOCK_PllClkSrc24M Pll clock source 24M.
kCLOCK_PllSrcClkPN Pll clock source CLK1_P and CLK1_N.

5.4.12 enum clock_pll_post_div_t

Enumerator

kCLOCK_PllPostDiv2 Divide by 2.
kCLOCK_PllPostDiv4 Divide by 4.
kCLOCK_PllPostDiv8 Divide by 8.
kCLOCK_PllPostDiv1 Divide by 1.

5.4.13 enum clock_pll_t

Enumerator

kCLOCK_PllArm ARM PLL.
kCLOCK_PllSys1 SYS1 PLL, it has a dedicated frequency of 1GHz.
kCLOCK_PllSys2 SYS2 PLL, it has a dedicated frequency of 528MHz.
kCLOCK_PllSys3 SYS3 PLL, it has a dedicated frequency of 480MHz.
kCLOCK_PllAudio Audio PLL.
kCLOCK_PllVideo Video PLL.
kCLOCK_PllInvalid Invalid value.

5.4.14 enum clock_pfd_t

Enumerator

kCLOCK_Pfd0 PLL PFD0.
kCLOCK_Pfd1 PLL PFD1.
kCLOCK_Pfd2 PLL PFD2.
kCLOCK_Pfd3 PLL PFD3.

5.4.15 enum clock_control_mode_t

Enumerator

kCLOCK_SoftwareMode Software control mode.
kCLOCK_GpcMode GPC control mode.

5.4.16 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.

5.4.17 enum clock_16MOsc_source_t

Enumerator

kCLOCK_16MOscSourceFrom16MOsc Source from 16MHz RC oscialltor.
kCLOCK_16MOscSourceFrom24MOsc Source from 24MHz crystal oscillator.

5.4.18 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.

5.4.19 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.

5.5 Function Documentation

5.5.1 static void CLOCK_SetRootClockMux (clock_root_t root, uint8_t src) [inline], [static]

Parameters

<i>root</i>	Which root clock node to set, see clock_root_t .
<i>src</i>	Clock mux value to set, different mux has different value range. See clock_root_mux_source_t .

5.5.2 static uint32_t CLOCK_GetRootClockMux (clock_root_t root) [inline], [static]

Parameters

<i>root</i>	Which root clock node to get, see clock_root_t .
-------------	--

Returns

Clock mux value.

5.5.3 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 clock_root_t .
<i>src</i>	Clock mux value to get, see clock_root_mux_source_t .

Returns

Clock source

5.5.4 static void CLOCK_SetRootClockDiv (clock_root_t *root*, uint8_t *div*) [inline], [static]

Parameters

<i>root</i>	Which root clock to set, see clock_root_t .
<i>div</i>	Clock div value to set, different divider has different value range.

5.5.5 static uint32_t CLOCK_GetRootClockDiv (clock_root_t *root*) [inline], [static]

Parameters

<i>root</i>	Which root clock node to get, see clock_root_t .
-------------	--

Returns

divider set for this root

5.5.6 `static void CLOCK_PowerOffRootClock (clock_root_t root) [inline],
[static]`

Parameters

<i>root</i>	Which root clock node to set, see clock_root_t .
-------------	--

5.5.7 static void CLOCK_PowerOnRootClock (clock_root_t *root*) [inline], [static]

Parameters

<i>root</i>	Which root clock node to set, see clock_root_t .
-------------	--

5.5.8 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 clock_root_t .
<i>config</i>	root clock config, see clock_root_config_t

5.5.9 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 clock_lpcg_t .
<i>value</i>	Clock gate value to set, see clock_gate_value_t .

5.5.10 static void CLOCK_EnableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

<i>name</i>	Which clock to enable, see clock_lpcg_t .
-------------	---

5.5.11 static void CLOCK_DisableClock (clock_ip_name_t *name*) [inline], [static]

Parameters

<i>name</i>	Which clock to disable, see clock_lpcg_t .
-------------	--

5.5.12 void CLOCK_SetGroupConfig (clock_group_t *group*, const clock_group_config_t * *config*)

Parameters

<i>group</i>	Which group to configure, see clock_group_t .
<i>config</i>	Configuration to set.

5.5.13 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 <code>clock_name_t</code>
-------------	--

Returns

Clock frequency value in hertz

5.5.14 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 <code>clock_root_t</code>
-------------	--

Returns

Clock frequency value in hertz

5.5.15 `static uint32_t CLOCK_GetM7Freq (void) [inline], [static]`

Returns

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

5.5.16 `static uint32_t CLOCK_GetM4Freq (void) [inline], [static]`

Returns

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

5.5.17 `static bool CLOCK_IsPllBypassed (clock_pll_t pll) [inline], [static]`

Parameters

<i>pll</i>	PLL control name (see clock_pll_t enumeration)
------------	--

Returns

PLL bypass status.

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

5.5.18 `static bool CLOCK_IsPllEnabled (clock_pll_t pll) [inline], [static]`

Parameters

<i>pll</i>	PLL control name (see clock_pll_t enumeration)
------------	--

Returns

PLL bypass status.

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

5.5.19 static uint32_t CLOCK_GetRtcFreq (void) [inline], [static]

Returns

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

5.5.20 static void CLOCK_OSC_SetOsc48MControlMode (clock_control_mode_t controlMode) [inline], [static]

Parameters

<i>controlMode</i>	The control mode to be set, please refer to clock_control_mode_t .
--------------------	--

5.5.21 static void CLOCK_OSC_EnableOsc48M (bool enable) [inline], [static]

Parameters

<i>enable</i>	Used to enable or disable the 48MHz RC oscillator. <ul style="list-style-type: none"> • true Enable the 48MHz RC oscillator. • false Dissable the 48MHz RC oscillator.
---------------	--

5.5.22 static void CLOCK_OSC_SetOsc48MDiv2ControlMode (clock_control_mode_t controlMode) [inline], [static]

Parameters

<i>controlMode</i>	The control mode to be set, please refer to clock_control_mode_t .
--------------------	--

5.5.23 static void CLOCK_OSC_EnableOsc48MDiv2 (bool *enable*) [inline], [static]

Note

The 48MHz RC oscillator must be enabled before enabling this 24MHz clock.

Parameters

<i>enable</i>	Used to enable/disable the 24MHz clock sourced from 48MHz RC oscillator. <ul style="list-style-type: none"> • true Enable the 24MHz clock sourced from 48MHz. • false Disable the 24MHz clock sourced from 48MHz.
---------------	---

5.5.24 static void CLOCK_OSC_SetOsc24MControlMode (clock_control_mode_t *controlMode*) [inline], [static]

Parameters

<i>controlMode</i>	The control mode to be set, please refer to clock_control_mode_t .
--------------------	--

5.5.25 void CLOCK_OSC_EnableOsc24M (void)

This function enables OSC 24Mhz.

5.5.26 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"> • true Gate the 24MHz crystal oscillator to save power. • false Ungate the 24MHz crystal oscillator.
-------------------	---

5.5.27 void CLOCK_OSC_SetOsc24MWorkMode (clock_24MOsc_mode_t workMode)

Parameters

<i>workMode</i>	The work mode of 24MHz crystal oscillator, please refer to clock_24MOsc_mode_t for details.
-----------------	---

5.5.28 static void CLOCK_OSC_SetOscRc400MControlMode (clock_control_mode_t controlMode) [inline], [static]

Parameters

<i>controlMode</i>	The control mode to be set, please refer to clock_control_mode_t .
--------------------	--

5.5.29 void CLOCK_OSC_EnableOscRc400M (void)

This function enables OSC RC 400Mhz.

5.5.30 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"> • true Gate the 400MHz RC oscillator. • false Ungate the 400MHz RC oscillator.
-------------------	---

5.5.31 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.

5.5.32 void CLOCK_OSC_SetOscRc400MRefClkDiv (uint8_t *divValue*)

Note

$\text{slow_clk} = \text{ref_clk} / (\text{divValue} + 1)$, and the recommend divide value is 24.

Parameters

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

5.5.33 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.
--------------------	--

5.5.34 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
----------------------	--

5.5.35 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"> • true Bypass the tune logic and use the programmed oscillator frequency to run the oscillator. Function CLOCK_OSC_SetOscRc400MTuneValue() can be used to set oscillator frequency. • false Use the output of tune logic to run the oscillator.
---------------------	--

5.5.36 void CLOCK_OSC_EnableOscRc400MTuneLogic (bool *enable*)

Parameters

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

5.5.37 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"> • true Freeze the tune at the current tuned value and the oscillator runs at the frozen tune value. • false Unfreezes and continues the tune operation.
---------------------	--

5.5.38 void CLOCK_OSC_SetOscRc400MTuneValue (uint8_t *tuneValue*)

Parameters

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

5.5.39 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 clock_1MHzOut_behavior_t for details.
-----------------	--

5.5.40 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.
--------------	--

5.5.41 bool CLOCK_OSC_CheckLocked1MHzErrorFlag (void)

Returns

The error flag for locked 1MHz clock out.

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

5.5.42 uint16_t CLOCK_OSC_GetCurrentOscRc400MFastClockCount (void)

Returns

The current count for the fast clock.

5.5.43 `uint8_t CLOCK_OSC_GetCurrentOscRc400MTuneValue (void)`

Returns

The current tune value.

5.5.44 `static void CLOCK_OSC_SetOsc16MControlMode (clock_control_mode_t controlMode) [inline], [static]`

Parameters

<i>controlMode</i>	The control mode to be set, please refer to clock_control_mode_t .
--------------------	--

5.5.45 `void CLOCK_OSC_SetOsc16MConfig (clock_16MOsc_source_t source, bool enablePowerSave, bool enableClockOut)`

Parameters

<i>source</i>	Used to select the source for 16MHz RC oscillator, please refer to clock_16MOsc_source_t .
<i>enablePower-Save</i>	Enable/disable power save mode function at 16MHz OSC. <ul style="list-style-type: none"> • true Enable power save mode function at 16MHz osc. • false Disable power save mode function at 16MHz osc.
<i>enableClock-Out</i>	Enable/Disable clock output for 16MHz RCOSC. <ul style="list-style-type: none"> • true Enable clock output for 16MHz RCOSC. • false Disable clock output for 16MHz RCOSC.

5.5.46 `void CLOCK_InitArmPll (const clock_arm_pll_config_t * config)`

This function initialize the ARM PLL with specific settings

Parameters

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

5.5.47 `status_t CLOCK_CalcArmPIIFreq (clock_arm_pll_config_t * config,
uint32_t freqInMhz)`

This function calculates config valudes per given frequency for Arm PLL

Parameters

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

5.5.48 **status_t** CLOCK_InitArmPllWithFreq (uint32_t *freqInMhz*)

This function initializes the Arm PLL with specific frequency

Parameters

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

5.5.49 **void** CLOCK_CalcPllSpreadSpectrum (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/video/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

5.5.50 **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.
---------------	-------------------------------

5.5.51 **void** CLOCK_GPC_SetSysPll1OutputFreq (const clock_sys_pll1_gpc_config_t * *config*)

Parameters

<i>config</i>	Pointer to clock_sys_pll1_gpc_config_t .
---------------	--

5.5.52 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 enabled spread spectrum
---------------	---

5.5.53 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.

5.5.54 void CLOCK_InitSysPll3 (void)

This function initializes the System PLL3 with specific settings

5.5.55 bool CLOCK_IsSysPll3PfdEnabled (clock_pfd_t *pfd*)

Parameters

<i>pfid</i>	PFD control name
-------------	------------------

Returns

PFD bypass status.

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

Note

Only useful in software control mode.

5.5.56 void CLOCK_SetPllBypass (clock_pll_t *pll*, bool *bypass*)

Parameters

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

5.5.57 status_t CLOCK_CalcAvPllFreq (clock_av_pll_config_t * *config*, uint32_t *freqInMhz*)

This function calculates config valudes per given frequency for Audio/Video PLL.

Parameters

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

5.5.58 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

5.5.59 void CLOCK_InitAudioPll (const clock_audio_pll_config_t * config)

This function initializes the Audio PLL with specific settings

Parameters

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

5.5.60 void CLOCK_GPC_SetAudioPllOutputFreq (const clock_audio_pll_gpc_config_t * config)

Parameters

<i>config</i>	Pointer to clock_audio_pll_gpc_config_t structure.
---------------	--

5.5.61 status_t CLOCK_InitVideoPllWithFreq (uint32_t freqInMhz, bool ssEnable, uint32_t ssRange, uint32_t ssMod)

This function initializes the Video 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

5.5.62 void CLOCK_InitVideoPll (const clock_video_pll_config_t * config)

This function configures the Video PLL with specific settings

Parameters

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

5.5.63 void CLOCK_GPC_SetVideoPllOutputFreq (const clock_video_pll_gpc_config_t * *config*)

Parameters

<i>config</i>	Pointer to clock_audio_pll_gpc_config_t structure.
---------------	--

5.5.64 uint32_t CLOCK_GetPllFreq (clock_pll_t *pll*)

This function get current output frequency of specific PLL

Parameters

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

Returns

The PLL output frequency in hertz.

5.5.65 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>pfd</i>	Which PFD clock to enable.
<i>frac</i>	The PFD FRAC value.

Note

It is recommended that PFD settings are kept between 12-35.

5.5.66 void CLOCK_DeinitPfd (clock_pll_t *pll*, clock_pfd_t *pfd*)

Parameters

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

5.5.67 uint32_t CLOCK_GetPfdFreq (clock_pll_t *pll*, clock_pfd_t *pfid*)

This function get current output frequency of specific System PLL PFD

Parameters

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

Returns

The PFD output frequency in hertz.

5.5.68 bool CLOCK_EnableUsbhs0Clock (clock_usb_src_t *src*, uint32_t *freq*)

This function only enables the access to USB HS peripheral, 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 kCLOCK_UsbSrc_Used .
<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.

5.5.69 bool CLOCK_EnableUsbhs1Clock (clock_usb_src_t *src*, uint32_t *freq*)

This function only enables the access to USB HS peripheral, 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 kCLOCK_UsbSrcUnused .
<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.

5.5.70 **bool CLOCK_EnableUsbhs0PhyPllClock (clock_usb_phy_src_t *src*, uint32_t *freq*)**

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

Parameters

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

Return values

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

5.5.71 **void CLOCK_DisableUsbhs0PhyPllClock (void)**

This function disables USB HS PHY PLL clock.

5.5.72 **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.

5.5.73 void CLOCK_DisableUsbhs1PhyPlIIClock (void)

This function disables USB HS PHY PLL clock.

**5.5.74 static void CLOCK_OSCPLL_LockControlMode (clock_name_t name)
[inline], [static]**

Note

When this bit is set, bits 16-20 can not be changed until next system reset.

Parameters

<i>name</i>	Clock source name, see clock_name_t .
-------------	---

**5.5.75 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 clock_name_t .
-------------	---

**5.5.76 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 clock_name_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.

5.5.77 static bool CLOCK_OSCPLL_IsSetPointImplemented (clock_name_t *name*) [inline], [static]

Parameters

<i>name</i>	Clock source name, see clock_name_t .
-------------	---

Returns

Clock source SetPoint implement status.

- true: SetPoint is implemented.
- false: SetPoint is not implemented.

5.5.78 static void CLOCK_OSCPLL_ControlByUnassignedMode (clock_name_t *name*) [inline], [static]

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 clock_name_t .
-------------	---

5.5.79 void CLOCK_OSCPLL_ControlBySetPointMode (clock_name_t *name*, uint16_t *spValue*, uint16_t *stbyValue*)

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 clock_name_t .
<i>spValue</i>	Bit0~Bit15 hold value for Setpoint 0~16 respectively. A bitfield value of 0 implies clock will be shutdown in this Setpoint. A bitfield value of 1 implies clock will be turn on in this Setpoint.
<i>stbyValue</i>	Bit0~Bit15 hold value for Setpoint 0~16 standby. A bitfield value of 0 implies clock will be shutdown during standby. A bitfield value of 1 represent clock will keep Setpoint setting during standby.

5.5.80 void CLOCK_OSCPLL_ControlByCpuLowPowerMode (clock_name_t *name*, uint8_t *domainId*, clock_level_t *level0*, clock_level_t *level1*)

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 clock_name_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.
<i>level0,level1</i>	Depend level of this clock.

5.5.81 static void CLOCK_OSCPLL_SetCurrentClockLevel (clock_name_t *name*, clock_level_t *level*) [inline], [static]

Note

This setting only take effects in CPU Low Power Mode.

Parameters

<i>name</i>	Clock source name, see clock_name_t .
<i>level</i>	Depend level of this clock.

5.5.82 static void CLOCK_OSCPLL_ControlByDomainMode (clock_name_t *name*, uint8_t *domainId*) [inline], [static]

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 clock_name_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.

5.5.83 static void CLOCK_ROOT_LockControlMode (clock_root_t *name*) [inline], [static]

Note

When this bit is set, bits 16-20 can not be changed until next system reset.

Parameters

<i>name</i>	Clock root name, see clock_root_t .
-------------	---

5.5.84 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 clock_root_t .
-------------	---

5.5.85 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 clock_root_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.

5.5.86 static bool CLOCK_ROOT_IsSetPointImplemented (clock_root_t *name*) [inline], [static]

Parameters

<i>name</i>	Clock root name, see clock_root_t .
-------------	---

Returns

Clock root SetPoint implement status.

- true: SetPoint is implemented.
- false: SetPoint is not implemented.

5.5.87 static void CLOCK_ROOT_ControlByUnassignedMode (clock_root_t *name*) [inline], [static]

Note

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

Parameters

<i>name</i>	Clock root name, see clock_root_t .
-------------	---

5.5.88 static void CLOCK_ROOT_ConfigSetPoint (clock_root_t *name*, uint16_t *splIndex*, const clock_root_setpoint_config_t * *config*) [inline], [static]

Note

SetPoint value could only be changed in Unassigend Mode.

Parameters

<i>name</i>	Which clock root to set, see clock_root_t .
<i>spIndex</i>	Which SetPoint of this clock root to set.
<i>config</i>	SetPoint config, see clock_root_setpoint_config_t

5.5.89 static void CLOCK_ROOT_EnableSetPointControl (clock_root_t *name*) [inline], [static]

Note

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

Parameters

<i>name</i>	Clock root name, see clock_root_t .
-------------	---

5.5.90 void CLOCK_ROOT_ControlBySetPointMode (clock_root_t *name*, const clock_root_setpoint_config_t * *spTable*)

Note

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

Parameters

<i>name</i>	Clock root name, see clock_root_t .
<i>spTable</i>	Point to the array that stores clock root settings for each setpoint. Note that the pointed array must have 16 elements.

5.5.91 static void CLOCK_ROOT_ControlByDomainMode (clock_root_t *name*, uint8_t *domainId*) [inline], [static]

Note

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

Parameters

<i>name</i>	Clock root name, see clock_root_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.

5.5.92 static void CLOCK_LPCG_LockControlMode (clock_lpcg_t *name*) [inline], [static]

Note

When this bit is set, bits 16-20 can not be changed until next system reset.

Parameters

<i>name</i>	Clock gate name, see clock_lpcg_t .
-------------	---

5.5.93 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 clock_lpcg_t .
-------------	---

5.5.94 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 clock_lpcg_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.

5.5.95 `static bool CLOCK_LPCG_IsSetPointImplemented (clock_lpcg_t name)`
`[inline], [static]`

Parameters

<i>name</i>	Clock gate name, see clock_lpcg_t .
-------------	---

Returns

Clock gate SetPoint implement status.

- true: SetPoint is implemented.
- false: SetPoint is not implemented.

5.5.96 static void CLOCK_LPCG_ControlByUnassignedMode (clock_lpcg_t *name*) [inline], [static]

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 clock_lpcg_t .
-------------	---

5.5.97 void CLOCK_LPCG_ControlBySetPointMode (clock_lpcg_t *name*, uint16_t *spValue*, uint16_t *stbyValue*)

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 clock_lpcg_t .
<i>spValue</i>	Bit0~Bit15 hold value for Setpoint 0~16 respectively. A bitfield value of 0 implies clock will be shutdown in this Setpoint. A bitfield value of 1 implies clock will be turn on in this Setpoint.
<i>stbyValue</i>	Bit0~Bit15 hold value for Setpoint 0~16 standby. A bitfield value of 0 implies clock will be shutdown during standby. A bitfield value of 1 represent clock will keep Setpoint setting during standby.

5.5.98 void CLOCK_LPCG_ControlByCpuLowPowerMode (clock_lpcg_t *name*, uint8_t *domainId*, clock_level_t *level0*, clock_level_t *level1*)

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 clock_lpcg_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.
<i>level0,level1</i>	Depend level of this clock.

5.5.99 static void CLOCK_LPCG_SetCurrentClockLevel (clock_lpcg_t *name*, clock_level_t *level*) [inline], [static]

Note

This setting only take effects in CPU Low Power Mode.

Parameters

<i>name</i>	Clock gate name, see clock_lpcg_t .
<i>level</i>	Depend level of this clock.

5.5.100 static void CLOCK_LPCG_ControlByDomainMode (clock_lpcg_t *name*, uint8_t *domainId*) [inline], [static]

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 clock_lpcg_t .
<i>domainId</i>	Domains that on the whitelist can change this clock.

Chapter 6

DCDC_SOC driver

6.1 Overview

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

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...](#)
- struct [dcdc_low_power_config_t](#)
Configuration for DCDC low power. [More...](#)
- struct [dcdc_setpoint_config_t](#)
DCDC configuration in set point mode. [More...](#)

Macros

- #define [FSL_DCDC_DRIVER_VERSION](#) (MAKE_VERSION(2, 1, 1))
DCDC driver version.
- #define [STANDBY_MODE_VDD1P0_TARGET_VOLTAGE](#)
The array of VDD1P0 target voltage in standby mode.

- #define `STANDBY_MODE_VDD1P8_TARGET_VOLTAGE`
The array of VDD1P8 target voltage in standby mode.
- #define `BUCK_MODE_VDD1P0_TARGET_VOLTAGE`
The array of VDD1P0 target voltage in buck mode.
- #define `BUCK_MODE_VDD1P8_TARGET_VOLTAGE`
The array of VDD1P8 target voltage in buck mode.

Enumerations

- enum `_dcdc_status_flags` { `kDCDC_AlreadySettledStatusFlag` = `DCDC_REG0_STS_DC_OK_M-ASK` }
The enumeration of DCDC status flags.
- enum `_dcdc_setpoint_map` {
`kDCDC_SetPoint0` = `1UL << 0UL`,
`kDCDC_SetPoint1` = `1UL << 1UL`,
`kDCDC_SetPoint2` = `1UL << 2UL`,
`kDCDC_SetPoint3` = `1UL << 3UL`,
`kDCDC_SetPoint4` = `1UL << 4UL`,
`kDCDC_SetPoint5` = `1UL << 5UL`,
`kDCDC_SetPoint6` = `1UL << 6UL`,
`kDCDC_SetPoint7` = `1UL << 7UL`,
`kDCDC_SetPoint8` = `1UL << 8UL`,
`kDCDC_SetPoint9` = `1UL << 9UL`,
`kDCDC_SetPoint10` = `1UL << 10UL`,
`kDCDC_SetPoint11` = `1UL << 11UL`,
`kDCDC_SetPoint12` = `1UL << 12UL`,
`kDCDC_SetPoint13` = `1UL << 13UL`,
`kDCDC_SetPoint14` = `1UL << 14UL`,
`kDCDC_SetPoint15` = `1UL << 15UL` }
System setpoints enumeration.
- enum `dcdc_control_mode_t` {
`kDCDC_StaticControl` = `0U`,
`kDCDC_SetPointControl` = `1U` }
DCDC control mode, including setpoint control mode and static 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_standby_mode_1P0_target_vol_t` {

```

kDCDC_1P0StbyTarget0P625V = 0U,
kDCDC_1P0StbyTarget0P65V,
kDCDC_1P0StbyTarget0P675V,
kDCDC_1P0StbyTarget0P7V,
kDCDC_1P0StbyTarget0P725V,
kDCDC_1P0StbyTarget0P75V,
kDCDC_1P0StbyTarget0P775V,
kDCDC_1P0StbyTarget0P8V,
kDCDC_1P0StbyTarget0P825V,
kDCDC_1P0StbyTarget0P85V,
kDCDC_1P0StbyTarget0P875V,
kDCDC_1P0StbyTarget0P9V,
kDCDC_1P0StbyTarget0P925V,
kDCDC_1P0StbyTarget0P95V,
kDCDC_1P0StbyTarget0P975V,
kDCDC_1P0StbyTarget1P0V,
kDCDC_1P0StbyTarget1P025V,
kDCDC_1P0StbyTarget1P05V,
kDCDC_1P0StbyTarget1P075V,
kDCDC_1P0StbyTarget1P1V,
kDCDC_1P0StbyTarget1P125V,
kDCDC_1P0StbyTarget1P15V,
kDCDC_1P0StbyTarget1P175V,
kDCDC_1P0StbyTarget1P2V,
kDCDC_1P0StbyTarget1P225V,
kDCDC_1P0StbyTarget1P25V,
kDCDC_1P0StbyTarget1P275V,
kDCDC_1P0StbyTarget1P3V,
kDCDC_1P0StbyTarget1P325V,
kDCDC_1P0StbyTarget1P35V,
kDCDC_1P0StbyTarget1P375V,
kDCDC_1P0StbyTarget1P4V = 0x1FU }

```

The enumeration VDD1P0's target voltage value in standby mode.

- enum `dcdc_standby_mode_1P8_target_vol_t` {


```

kDCDC_1P8StbyTarget1P525V = 0U,
kDCDC_1P8StbyTarget1P55V,
kDCDC_1P8StbyTarget1P575V,
kDCDC_1P8StbyTarget1P6V,
kDCDC_1P8StbyTarget1P625V,
kDCDC_1P8StbyTarget1P65V,
kDCDC_1P8StbyTarget1P675V,
kDCDC_1P8StbyTarget1P7V,
kDCDC_1P8StbyTarget1P725V,
kDCDC_1P8StbyTarget1P75V,
kDCDC_1P8StbyTarget1P775V,
kDCDC_1P8StbyTarget1P8V,
kDCDC_1P8StbyTarget1P825V,
kDCDC_1P8StbyTarget1P85V,
kDCDC_1P8StbyTarget1P875V,
kDCDC_1P8StbyTarget1P9V,
kDCDC_1P8StbyTarget1P925V,
kDCDC_1P8StbyTarget1P95V,
kDCDC_1P8StbyTarget1P975V,
kDCDC_1P8StbyTarget2P0V,
kDCDC_1P8StbyTarget2P025V,
kDCDC_1P8StbyTarget2P05V,
kDCDC_1P8StbyTarget2P075V,
kDCDC_1P8StbyTarget2P1V,
kDCDC_1P8StbyTarget2P125V,
kDCDC_1P8StbyTarget2P15V,
kDCDC_1P8StbyTarget2P175V,
kDCDC_1P8StbyTarget2P2V,
kDCDC_1P8StbyTarget2P225V,
kDCDC_1P8StbyTarget2P25V,
kDCDC_1P8StbyTarget2P275V,
kDCDC_1P8StbyTarget2P3V = 0x1FU }

```

The enumeration VDD1P8's target voltage value in standby mode.

- enum `dcdc_buck_mode_1P0_target_vol_t` {

```

kDCDC_1P0BuckTarget0P6V = 0U,
kDCDC_1P0BuckTarget0P625V,
kDCDC_1P0BuckTarget0P65V,
kDCDC_1P0BuckTarget0P675V,
kDCDC_1P0BuckTarget0P7V,
kDCDC_1P0BuckTarget0P725V,
kDCDC_1P0BuckTarget0P75V,
kDCDC_1P0BuckTarget0P775V,
kDCDC_1P0BuckTarget0P8V,
kDCDC_1P0BuckTarget0P825V,
kDCDC_1P0BuckTarget0P85V,
kDCDC_1P0BuckTarget0P875V,
kDCDC_1P0BuckTarget0P9V,
kDCDC_1P0BuckTarget0P925V,
kDCDC_1P0BuckTarget0P95V,
kDCDC_1P0BuckTarget0P975V,
kDCDC_1P0BuckTarget1P0V,
kDCDC_1P0BuckTarget1P025V,
kDCDC_1P0BuckTarget1P05V,
kDCDC_1P0BuckTarget1P075V,
kDCDC_1P0BuckTarget1P1V,
kDCDC_1P0BuckTarget1P125V,
kDCDC_1P0BuckTarget1P15V,
kDCDC_1P0BuckTarget1P175V,
kDCDC_1P0BuckTarget1P2V,
kDCDC_1P0BuckTarget1P225V,
kDCDC_1P0BuckTarget1P25V,
kDCDC_1P0BuckTarget1P275V,
kDCDC_1P0BuckTarget1P3V,
kDCDC_1P0BuckTarget1P325V,
kDCDC_1P0BuckTarget1P35V,
kDCDC_1P0BuckTarget1P375V = 0x1FU }

```

The enumeration VDD1P0's target voltage value in buck mode.

- enum `dcdc_buck_mode_1P8_target_vol_t` {

```

kDCDC_1P8BuckTarget1P5V = 0U,
kDCDC_1P8BuckTarget1P525V,
kDCDC_1P8BuckTarget1P55V,
kDCDC_1P8BuckTarget1P575V,
kDCDC_1P8BuckTarget1P6V,
kDCDC_1P8BuckTarget1P625V,
kDCDC_1P8BuckTarget1P65V,
kDCDC_1P8BuckTarget1P675V,
kDCDC_1P8BuckTarget1P7V,
kDCDC_1P8BuckTarget1P725V,
kDCDC_1P8BuckTarget1P75V,
kDCDC_1P8BuckTarget1P775V,
kDCDC_1P8BuckTarget1P8V,
kDCDC_1P8BuckTarget1P825V,
kDCDC_1P8BuckTarget1P85V,
kDCDC_1P8BuckTarget1P875V,
kDCDC_1P8BuckTarget1P9V,
kDCDC_1P8BuckTarget1P925V,
kDCDC_1P8BuckTarget1P95V,
kDCDC_1P8BuckTarget1P975V,
kDCDC_1P8BuckTarget2P0V,
kDCDC_1P8BuckTarget2P025V,
kDCDC_1P8BuckTarget2P05V,
kDCDC_1P8BuckTarget2P075V,
kDCDC_1P8BuckTarget2P1V,
kDCDC_1P8BuckTarget2P125V,
kDCDC_1P8BuckTarget2P15V,
kDCDC_1P8BuckTarget2P175V,
kDCDC_1P8BuckTarget2P2V,
kDCDC_1P8BuckTarget2P225V,
kDCDC_1P8BuckTarget2P25V,
kDCDC_1P8BuckTarget2P275V = 0x1FU }

```

The enumeration VDD1P8's target voltage value in buck mode.

- 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.

Power Mode Related Interfaces

- static void `DCDC_EnterLowPowerModeViaStandbyRequest` (DCDC_Type *base, bool enable)
Makes the DCDC enter into low power mode for GPC standby request or not.
- static void `DCDC_EnterLowPowerMode` (DCDC_Type *base, bool enable)
Makes DCDC enter into low power mode or not, before entering low power mode must disable stepping for VDD1P8 and VDD1P0.
- static void `DCDC_EnterStandbyMode` (DCDC_Type *base, bool enable)
Makes DCDC enter into standby mode or not.

Outputs' Target Voltage Related Interfaces

- static void `DCDC_SetVDD1P0StandbyModeTargetVoltage` (DCDC_Type *base, `dcdc_standby_mode_1P0_target_vol_t` targetVoltage)
Sets the target value(ranges from 0.625V to 1.4V) of VDD1P0 in standby mode, 25mV each step.
- static uint16_t `DCDC_GetVDD1P0StandbyModeTargetVoltage` (DCDC_Type *base)
Gets the target value of VDD1P0 in standby mode, the result takes "mV" as the unit.
- static void `DCDC_SetVDD1P8StandbyModeTargetVoltage` (DCDC_Type *base, `dcdc_standby_mode_1P8_target_vol_t` targetVoltage)
Sets the target value(ranges from 1.525V to 2.3V) of VDD1P8 in standby mode, 25mV each step.
- static uint16_t `DCDC_GetVDD1P8StandbyModeTargetVoltage` (DCDC_Type *base)
Gets the target value of VDD1P8 in standby mode, the result takes "mV" as the unit.
- static void `DCDC_SetVDD1P0BuckModeTargetVoltage` (DCDC_Type *base, `dcdc_buck_mode_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_SetVDD1P8BuckModeTargetVoltage` (DCDC_Type *base, `dcdc_buck_mode_1P8_target_vol_t` targetVoltage)
Sets the target value(ranges from 1.5V to 2.275V) of VDD1P8 in buck mode, 25mV each step.

- static uint16_t [DCDC_GetVDD1P8BuckModeTargetVoltage](#) (DCDC_Type *base)
Gets the target value of VDD1P8 in buck mode, 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.
- static void [DCDC_EnableVDD1P8TargetVoltageStepping](#) (DCDC_Type *base, bool enable)
Enables/Disables stepping for VDD1P8, before entering low power modes the stepping for VDD1P8 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

- static void [DCDC_EnableOutputRangeComparator](#) (DCDC_Type *base, bool enable)
Enables/Disables the output range comparator.
- void [DCDC_SetClockSource](#) (DCDC_Type *base, dcdc_clock_source_t clockSource)
Configures the DCDC clock source.
- void [DCDC_GetDefaultLowPowerConfig](#) (dcdc_low_power_config_t *config)
Gets the default setting for low power configuration.
- void [DCDC_SetLowPowerConfig](#) (DCDC_Type *base, const dcdc_low_power_config_t *config)
Configures the DCDC low power.
- static void [DCDC_SetBandgapVoltageTrimValue](#) (DCDC_Type *base, uint32_t trimValue)
Sets the bandgap 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_SetMinPowerConfig](#) (DCDC_Type *base, const dcdc_min_power_config_t *config)
Configures for the min power.
- static void [DCDC_SetLPComparatorBiasValue](#) (DCDC_Type *base, dcdc_comparator_current_bias_t biasValue)
Sets the current bias of low power comparator.
- 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.
- static void [DCDC_EnableImproveTransition](#) (DCDC_Type *base, bool enable)
Enables/Disables to improve the transition from heavy load to light load.

Setpoint Control Related Interfaces

- void [DCDC_SetPointInit](#) (DCDC_Type *base, const dcdc_setpoint_config_t *config)
Initializes DCDC module when the control mode selected as setpoint mode.

- static void [DCDC_SetPointDeinit](#) (DCDC_Type *base, uint32_t setpointMap)
Disable DCDC module when the control mode selected as setpoint mode.

DCDC Status Related Interfaces

- static uint32_t [DCDC_GetStatusFlags](#) (DCDC_Type *base)
Get DCDC status flags.

Application Guideline Interfaces

- void [DCDC_BootIntoDCM](#) (DCDC_Type *base)
Boots DCDC into DCM(discontinous conduction mode).
- void [DCDC_BootIntoCCM](#) (DCDC_Type *base)
Boots DCDC into CCM(continous conduction mode).

6.2 Data Structure Documentation

6.2.1 struct dc_dc_config_t

Data Fields

- [dc_dc_control_mode_t controlMode](#)
DCDC control mode.
- [dc_dc_trim_input_mode_t trimInputMode](#)
Hold trim input.
- bool [enableDcdcTimeout](#)
Enable internal count for DCDC_OK timeout.
- bool [enableSwitchingConverterOutput](#)
Enable the VDDIO switching converter output.

Field Documentation

- (1) [dc_dc_control_mode_t dc_dc_config_t::controlMode](#)
- (2) [dc_dc_trim_input_mode_t dc_dc_config_t::trimInputMode](#)
- (3) [bool dc_dc_config_t::enableDcdcTimeout](#)
- (4) [bool dc_dc_config_t::enableSwitchingConverterOutput](#)

6.2.2 struct dc_dc_min_power_config_t

Data Fields

- bool [enableUseHalfFreqForContinuous](#)
Set DCDC clock to half frequency for the continuous mode.

Field Documentation

(1) `bool dc_dc_min_power_config_t::enableUseHalfFreqForContinuous`

6.2.3 struct `dc_dc_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.
- `bool powerDownPeakCurrentDetection`
Power down peak-current detection.
- `bool powerDownZeroCrossDetection`
Power down the zero cross detection function for discontinuous conductor mode.
- `dc_dc_peak_current_threshold_t PeakCurrentThreshold`
The threshold of peak current detection.

Field Documentation

- (1) `bool dc_dc_detection_config_t::enableXtalokDetection`
- (2) `bool dc_dc_detection_config_t::powerDownOverVoltageVdd1P8Detection`
- (3) `bool dc_dc_detection_config_t::powerDownOverVoltageVdd1P0Detection`
- (4) `bool dc_dc_detection_config_t::powerDownLowVoltageDetection`
- (5) `bool dc_dc_detection_config_t::powerDownOverCurrentDetection`
- (6) `bool dc_dc_detection_config_t::powerDownPeakCurrentDetection`
- (7) `bool dc_dc_detection_config_t::powerDownZeroCrossDetection`
- (8) `dc_dc_peak_current_threshold_t dc_dc_detection_config_t::PeakCurrentThreshold`

6.2.4 struct `dc_dc_loop_control_config_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 `dcdc_loop_control_config_t::enableCommonHysteresis`

This feature will improve transient supply ripple and efficiency.

(2) bool `dcdc_loop_control_config_t::enableCommonThresholdDetection`

(3) bool `dcdc_loop_control_config_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 dc_dc_loop_control_config_t::integralProportionalRatio

6.2.5 struct dc_dc_internal_regulator_config_t

Data Fields

- uint32_t [feedbackPoint](#)
Available range is 0~3.

Field Documentation

(1) uint32_t dc_dc_internal_regulator_config_t::feedbackPoint

Select the feedback point of the internal regulator.

6.2.6 struct dc_dc_low_power_config_t

Data Fields

- bool [enableAdjustHystereticValue](#)
Adjust hysteretic value in low power from 12.5mV to 25mV.

Field Documentation

(1) bool dc_dc_low_power_config_t::enableAdjustHystereticValue

6.2.7 struct dc_dc_setpoint_config_t

Data Fields

- uint32_t [enableDCDCMap](#)
The setpoint map that enable the DCDC module.
- uint32_t [enableDigLogicMap](#)
The setpoint map that enable the DCDC dig logic.
- uint32_t [lowpowerMap](#)
The setpoint map that enable the DCDC Low powermode.
- uint32_t [standbyMap](#)
The setpoint map that enable the DCDC standby mode.
- uint32_t [standbyLowpowerMap](#)
The setpoint map that enable the DCDC low power mode, when the related setpoint is in standby mode.
- [dc_dc_buck_mode_1P8_target_vol_t](#) * [buckVDD1P8TargetVoltage](#)
Point to the array that store the target voltage level of VDD1P8 in buck mode, please refer to [dc_dc_buck-mode_1P8_target_vol_t](#).
- [dc_dc_buck_mode_1P0_target_vol_t](#) * [buckVDD1P0TargetVoltage](#)
Point to the array that store the target voltage level of VDD1P0 in buck mode, please refer to [dc_dc_buck-mode_1P0_target_vol_t](#).
- [dc_dc_standby_mode_1P8_target_vol_t](#) * [standbyVDD1P8TargetVoltage](#)

- Point to the array that store the target
voltage level of VDD1P8 in standby mode, please refer to [dcdc_standby_mode_1P8_target_vol_t](#).
- [dcdc_standby_mode_1P0_target_vol_t](#) * [standbyVDD1P0TargetVoltage](#)
Point to the array that store the target
voltage level of VDD1P0 in standby mode, please refer to [dcdc_standby_mode_1P0_target_vol_t](#).

Field Documentation

(1) `uint32_t dcdc_setpoint_config_t::enableDCDCMap`

Should be the OR'ed value of [_dcdc_setpoint_map](#).

(2) `uint32_t dcdc_setpoint_config_t::enableDigLogicMap`

Should be the OR'ed value of [_dcdc_setpoint_map](#).

(3) `uint32_t dcdc_setpoint_config_t::lowpowerMap`

Should be the OR'ed value of [_dcdc_setpoint_map](#).

(4) `uint32_t dcdc_setpoint_config_t::standbyMap`

Should be the OR'ed value of [_dcdc_setpoint_map](#).

(5) `uint32_t dcdc_setpoint_config_t::standbyLowpowerMap`

Please refer to [_dcdc_setpoint_map](#).

(6) `dc_dc_buck_mode_1P8_target_vol_t* dcdc_setpoint_config_t::buckVDD1P8TargetVoltage`

Note that the pointed array must have 16 elements.

(7) `dc_dc_buck_mode_1P0_target_vol_t* dcdc_setpoint_config_t::buckVDD1P0TargetVoltage`

Note that the pointed array must have 16 elements.

(8) `dc_dc_standby_mode_1P8_target_vol_t* dcdc_setpoint_config_t::standbyVDD1P8TargetVoltage`

Note that the pointed array must have 16 elements.

(9) `dc_dc_standby_mode_1P0_target_vol_t* dcdc_setpoint_config_t::standbyVDD1P0TargetVoltage`

Note that the pointed array must have 16 elements.

6.3 Macro Definition Documentation

6.3.1 #define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Version 2.1.1.

6.3.2 #define STANDBY_MODE_VDD1P0_TARGET_VOLTAGE**Value:**

```
{
    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, 1400
}
```

6.3.3 #define STANDBY_MODE_VDD1P8_TARGET_VOLTAGE**Value:**

```
{
    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, 2300
}
```

6.3.4 #define BUCK_MODE_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
}
```

6.3.5 #define BUCK_MODE_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
    \
}
```

6.4 Enumeration Type Documentation

6.4.1 enum _dcdc_status_flags

Enumerator

kDCDC_AlreadySettledStatusFlag Indicate DCDC status. 1'b1: DCDC already settled 1'b0: DC-DC is settling.

6.4.2 enum _dcdc_setpoint_map

Enumerator

kDCDC_SetPoint0 Set point 0.
kDCDC_SetPoint1 Set point 1.
kDCDC_SetPoint2 Set point 2.
kDCDC_SetPoint3 Set point 3.
kDCDC_SetPoint4 Set point 4.
kDCDC_SetPoint5 Set point 5.
kDCDC_SetPoint6 Set point 6.
kDCDC_SetPoint7 Set point 7.
kDCDC_SetPoint8 Set point 8.
kDCDC_SetPoint9 Set point 9.
kDCDC_SetPoint10 Set point 10.
kDCDC_SetPoint11 Set point 11.
kDCDC_SetPoint12 Set point 12.
kDCDC_SetPoint13 Set point 13.
kDCDC_SetPoint14 Set point 14.
kDCDC_SetPoint15 Set point 15.

6.4.3 enum dcdc_control_mode_t

Enumerator

kDCDC_StaticControl Static control.
kDCDC_SetPointControl Controlled by GPC set points.

6.4.4 enum dcdc_trim_input_mode_t

Enumerator

kDCDC_SampleTrimInput Sample trim input.

kDCDC_HoldTrimInput Hold trim input.

6.4.5 enum dcdc_standby_mode_1P0_target_vol_t

Enumerator

kDCDC_1P0StbyTarget0P625V In standby mode, the target voltage value of VDD1P0 is 0.625V.

kDCDC_1P0StbyTarget0P65V In standby mode, the target voltage value of VDD1P0 is 0.65V.

kDCDC_1P0StbyTarget0P675V In standby mode, the target voltage value of VDD1P0 is 0.675V.

kDCDC_1P0StbyTarget0P7V In standby mode, the target voltage value of VDD1P0 is 0.7V.

kDCDC_1P0StbyTarget0P725V In standby mode, the target voltage value of VDD1P0 is 0.725V.

kDCDC_1P0StbyTarget0P75V In standby mode, the target voltage value of VDD1P0 is 0.75V.

kDCDC_1P0StbyTarget0P775V In standby mode, the target voltage value of VDD1P0 is 0.775V.

kDCDC_1P0StbyTarget0P8V In standby mode, the target voltage value of VDD1P0 is 0.8V.

kDCDC_1P0StbyTarget0P825V In standby mode, the target voltage value of VDD1P0 is 0.825V.

kDCDC_1P0StbyTarget0P85V In standby mode, the target voltage value of VDD1P0 is 0.85V.

kDCDC_1P0StbyTarget0P875V In standby mode, the target voltage value of VDD1P0 is 0.875V.

kDCDC_1P0StbyTarget0P9V In standby mode, the target voltage value of VDD1P0 is 0.9V.

kDCDC_1P0StbyTarget0P925V In standby mode, the target voltage value of VDD1P0 is 0.925V.

kDCDC_1P0StbyTarget0P95V In standby mode, the target voltage value of VDD1P0 is 0.95V.

kDCDC_1P0StbyTarget0P975V In standby mode, the target voltage value of VDD1P0 is 0.975V.

kDCDC_1P0StbyTarget1P0V In standby mode, the target voltage value of VDD1P0 is 1.0V.

kDCDC_1P0StbyTarget1P025V In standby mode, the target voltage value of VDD1P0 is 1.025V.

kDCDC_1P0StbyTarget1P05V In standby mode, the target voltage value of VDD1P0 is 1.05V.

kDCDC_1P0StbyTarget1P075V In standby mode, the target voltage value of VDD1P0 is 1.075V.

kDCDC_1P0StbyTarget1P1V In standby mode, the target voltage value of VDD1P0 is 1.1V.

kDCDC_1P0StbyTarget1P125V In standby mode, the target voltage value of VDD1P0 is 1.125V.

kDCDC_1P0StbyTarget1P15V In standby mode, the target voltage value of VDD1P0 is 1.15V.

kDCDC_1P0StbyTarget1P175V In standby mode, the target voltage value of VDD1P0 is 1.175V.

kDCDC_1P0StbyTarget1P2V In standby mode, the target voltage value of VDD1P0 is 1.2V.

kDCDC_1P0StbyTarget1P225V In standby mode, the target voltage value of VDD1P0 is 1.225V.

kDCDC_1P0StbyTarget1P25V In standby mode, the target voltage value of VDD1P0 is 1.25V.

kDCDC_1P0StbyTarget1P275V In standby mode, the target voltage value of VDD1P0 is 1.275V.

kDCDC_1P0StbyTarget1P3V In standby mode, the target voltage value of VDD1P0 is 1.3V.

kDCDC_1P0StbyTarget1P325V In standby mode, the target voltage value of VDD1P0 is 1.325V.

kDCDC_1P0StbyTarget1P35V In standby mode, the target voltage value of VDD1P0 is 1.35V.

kDCDC_1P0StbyTarget1P375V In standby mode, the target voltage value of VDD1P0 is 1.375V.

kDCDC_1P0StbyTarget1P4V In standby mode, The target voltage value of VDD1P0 is 1.4V.

6.4.6 enum dcdc_standby_mode_1P8_target_vol_t

Enumerator

kDCDC_1P8StbyTarget1P525V In standby mode, the target voltage value of VDD1P8 is 1.525V.
kDCDC_1P8StbyTarget1P55V In standby mode, the target voltage value of VDD1P8 is 1.55V.
kDCDC_1P8StbyTarget1P575V In standby mode, the target voltage value of VDD1P8 is 1.575V.
kDCDC_1P8StbyTarget1P6V In standby mode, the target voltage value of VDD1P8 is 1.6V.
kDCDC_1P8StbyTarget1P625V In standby mode, the target voltage value of VDD1P8 is 1.625V.
kDCDC_1P8StbyTarget1P65V In standby mode, the target voltage value of VDD1P8 is 1.65V.
kDCDC_1P8StbyTarget1P675V In standby mode, the target voltage value of VDD1P8 is 1.675V.
kDCDC_1P8StbyTarget1P7V In standby mode, the target voltage value of VDD1P8 is 1.7V.
kDCDC_1P8StbyTarget1P725V In standby mode, the target voltage value of VDD1P8 is 1.725V.
kDCDC_1P8StbyTarget1P75V In standby mode, the target voltage value of VDD1P8 is 1.75V.
kDCDC_1P8StbyTarget1P775V In standby mode, the target voltage value of VDD1P8 is 1.775V.
kDCDC_1P8StbyTarget1P8V In standby mode, the target voltage value of VDD1P8 is 1.8V.
kDCDC_1P8StbyTarget1P825V In standby mode, the target voltage value of VDD1P8 is 1.825V.
kDCDC_1P8StbyTarget1P85V In standby mode, the target voltage value of VDD1P8 is 1.85V.
kDCDC_1P8StbyTarget1P875V In standby mode, the target voltage value of VDD1P8 is 1.875V.
kDCDC_1P8StbyTarget1P9V In standby mode, the target voltage value of VDD1P8 is 1.9V.
kDCDC_1P8StbyTarget1P925V In standby mode, the target voltage value of VDD1P8 is 1.925V.
kDCDC_1P8StbyTarget1P95V In standby mode, the target voltage value of VDD1P8 is 1.95V.
kDCDC_1P8StbyTarget1P975V In standby mode, the target voltage value of VDD1P8 is 1.975V.
kDCDC_1P8StbyTarget2P0V In standby mode, the target voltage value of VDD1P8 is 2.0V.
kDCDC_1P8StbyTarget2P025V In standby mode, the target voltage value of VDD1P8 is 2.025V.
kDCDC_1P8StbyTarget2P05V In standby mode, the target voltage value of VDD1P8 is 2.05V.
kDCDC_1P8StbyTarget2P075V In standby mode, the target voltage value of VDD1P8 is 2.075V.
kDCDC_1P8StbyTarget2P1V In standby mode, the target voltage value of VDD1P8 is 2.1V.
kDCDC_1P8StbyTarget2P125V In standby mode, the target voltage value of VDD1P8 is 2.125V.
kDCDC_1P8StbyTarget2P15V In standby mode, the target voltage value of VDD1P8 is 2.15V.
kDCDC_1P8StbyTarget2P175V In standby mode, the target voltage value of VDD1P8 is 2.175V.
kDCDC_1P8StbyTarget2P2V In standby mode, the target voltage value of VDD1P8 is 2.2V.
kDCDC_1P8StbyTarget2P225V In standby mode, the target voltage value of VDD1P8 is 2.225V.
kDCDC_1P8StbyTarget2P25V In standby mode, the target voltage value of VDD1P8 is 2.25V.
kDCDC_1P8StbyTarget2P275V In standby mode, the target voltage value of VDD1P8 is 2.275V.
kDCDC_1P8StbyTarget2P3V In standby mode, the target voltage value is 2.3V.

6.4.7 enum dcdc_buck_mode_1P0_target_vol_t

Enumerator

kDCDC_1P0BuckTarget0P6V In buck mode, the target voltage value of VDD1P0 is 0.6V.
kDCDC_1P0BuckTarget0P625V In buck mode, the target voltage value of VDD1P0 is 0.625V.
kDCDC_1P0BuckTarget0P65V In buck mode, the target voltage value of VDD1P0 is 0.65V.

kDCDC_1P0BuckTarget0P675V In buck mode, the target voltage value of VDD1P0 is 0.675V.
kDCDC_1P0BuckTarget0P7V In buck mode, the target voltage value of VDD1P0 is 0.7V.
kDCDC_1P0BuckTarget0P725V In buck mode, the target voltage value of VDD1P0 is 0.725V.
kDCDC_1P0BuckTarget0P75V In buck mode, the target voltage value of VDD1P0 is 0.75V.
kDCDC_1P0BuckTarget0P775V In buck mode, the target voltage value of VDD1P0 is 0.775V.
kDCDC_1P0BuckTarget0P8V In buck mode, the target voltage value of VDD1P0 is 0.8V.
kDCDC_1P0BuckTarget0P825V In buck mode, the target voltage value of VDD1P0 is 0.825V.
kDCDC_1P0BuckTarget0P85V In buck mode, the target voltage value of VDD1P0 is 0.85V.
kDCDC_1P0BuckTarget0P875V In buck mode, the target voltage value of VDD1P0 is 0.875V.
kDCDC_1P0BuckTarget0P9V In buck mode, the target voltage value of VDD1P0 is 0.9V.
kDCDC_1P0BuckTarget0P925V In buck mode, the target voltage value of VDD1P0 is 0.925V.
kDCDC_1P0BuckTarget0P95V In buck mode, the target voltage value of VDD1P0 is 0.95V.
kDCDC_1P0BuckTarget0P975V In buck mode, the target voltage value of VDD1P0 is 0.975V.
kDCDC_1P0BuckTarget1P0V In buck mode, the target voltage value of VDD1P0 is 1.0V.
kDCDC_1P0BuckTarget1P025V In buck mode, the target voltage value of VDD1P0 is 1.025V.
kDCDC_1P0BuckTarget1P05V In buck mode, the target voltage value of VDD1P0 is 1.05V.
kDCDC_1P0BuckTarget1P075V In buck mode, the target voltage value of VDD1P0 is 1.075V.
kDCDC_1P0BuckTarget1P1V In buck mode, the target voltage value of VDD1P0 is 1.1V.
kDCDC_1P0BuckTarget1P125V In buck mode, the target voltage value of VDD1P0 is 1.125V.
kDCDC_1P0BuckTarget1P15V In buck mode, the target voltage value of VDD1P0 is 1.15V.
kDCDC_1P0BuckTarget1P175V In buck mode, the target voltage value of VDD1P0 is 1.175V.
kDCDC_1P0BuckTarget1P2V In buck mode, the target voltage value of VDD1P0 is 1.2V.
kDCDC_1P0BuckTarget1P225V In buck mode, the target voltage value of VDD1P0 is 1.225V.
kDCDC_1P0BuckTarget1P25V In buck mode, the target voltage value of VDD1P0 is 1.25V.
kDCDC_1P0BuckTarget1P275V In buck mode, the target voltage value of VDD1P0 is 1.275V.
kDCDC_1P0BuckTarget1P3V In buck mode, the target voltage value of VDD1P0 is 1.3V.
kDCDC_1P0BuckTarget1P325V In buck mode, the target voltage value of VDD1P0 is 1.325V.
kDCDC_1P0BuckTarget1P35V In buck mode, the target voltage value of VDD1P0 is 1.35V.
kDCDC_1P0BuckTarget1P375V In buck mode, the target voltage value of VDD1P0 is 1.375V.

6.4.8 enum dcdc_buck_mode_1P8_target_vol_t

Enumerator

kDCDC_1P8BuckTarget1P5V In buck mode, the target voltage value of VDD1P0 is 1.5V.
kDCDC_1P8BuckTarget1P525V In buck mode, the target voltage value of VDD1P0 is 1.525V.
kDCDC_1P8BuckTarget1P55V In buck mode, the target voltage value of VDD1P0 is 1.55V.
kDCDC_1P8BuckTarget1P575V In buck mode, the target voltage value of VDD1P0 is 1.575V.
kDCDC_1P8BuckTarget1P6V In buck mode, the target voltage value of VDD1P0 is 1.6V.
kDCDC_1P8BuckTarget1P625V In buck mode, the target voltage value of VDD1P0 is 1.625V.
kDCDC_1P8BuckTarget1P65V In buck mode, the target voltage value of VDD1P0 is 1.65V.
kDCDC_1P8BuckTarget1P675V In buck mode, the target voltage value of VDD1P0 is 1.675V.
kDCDC_1P8BuckTarget1P7V In buck mode, the target voltage value of VDD1P0 is 1.7V.
kDCDC_1P8BuckTarget1P725V In buck mode, the target voltage value of VDD1P0 is 1.725V.

kDCDC_1P8BuckTarget1P75V In buck mode, the target voltage value of VDD1P0 is 1.75V.
kDCDC_1P8BuckTarget1P775V In buck mode, the target voltage value of VDD1P0 is 1.775V.
kDCDC_1P8BuckTarget1P8V In buck mode, the target voltage value of VDD1P0 is 1.8V.
kDCDC_1P8BuckTarget1P825V In buck mode, the target voltage value of VDD1P0 is 1.825V.
kDCDC_1P8BuckTarget1P85V In buck mode, the target voltage value of VDD1P0 is 1.85V.
kDCDC_1P8BuckTarget1P875V In buck mode, the target voltage value of VDD1P0 is 1.875V.
kDCDC_1P8BuckTarget1P9V In buck mode, the target voltage value of VDD1P0 is 1.9V.
kDCDC_1P8BuckTarget1P925V In buck mode, the target voltage value of VDD1P0 is 1.925V.
kDCDC_1P8BuckTarget1P95V In buck mode, the target voltage value of VDD1P0 is 1.95V.
kDCDC_1P8BuckTarget1P975V In buck mode, the target voltage value of VDD1P0 is 1.975V.
kDCDC_1P8BuckTarget2P0V In buck mode, the target voltage value of VDD1P0 is 2.0V.
kDCDC_1P8BuckTarget2P025V In buck mode, the target voltage value of VDD1P0 is 2.025V.
kDCDC_1P8BuckTarget2P05V In buck mode, the target voltage value of VDD1P0 is 2.05V.
kDCDC_1P8BuckTarget2P075V In buck mode, the target voltage value of VDD1P0 is 2.075V.
kDCDC_1P8BuckTarget2P1V In buck mode, the target voltage value of VDD1P0 is 2.1V.
kDCDC_1P8BuckTarget2P125V In buck mode, the target voltage value of VDD1P0 is 2.125V.
kDCDC_1P8BuckTarget2P15V In buck mode, the target voltage value of VDD1P0 is 2.15V.
kDCDC_1P8BuckTarget2P175V In buck mode, the target voltage value of VDD1P0 is 2.175V.
kDCDC_1P8BuckTarget2P2V In buck mode, the target voltage value of VDD1P0 is 2.2V.
kDCDC_1P8BuckTarget2P225V In buck mode, the target voltage value of VDD1P0 is 2.225V.
kDCDC_1P8BuckTarget2P25V In buck mode, the target voltage value of VDD1P0 is 2.25V.
kDCDC_1P8BuckTarget2P275V In buck mode, the target voltage value of VDD1P0 is 2.275V.

6.4.9 enum dc_dc_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.

6.4.10 enum dc_dc_peak_current_threshold_t

Enumerator

kDCDC_PeakCurrentRunMode250mALPMode1P5A Over peak current threshold in low power mode is 250mA, in run mode is 1.5A.
kDCDC_PeakCurrentRunMode200mALPMode1P5A Over peak current threshold in low power mode is 200mA, in run mode is 1.5A.
kDCDC_PeakCurrentRunMode250mALPMode2A Over peak current threshold in low power mode is 250mA, in run mode is 2A.

kDCDC_PeakCurrentRunMode200mALPMode2A Over peak current threshold in low power mode is 200mA, in run mode is 2A.

6.4.11 enum dcdc_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.

6.4.12 enum dcdc_voltage_output_sel_t

Enumerator

kDCDC_VoltageOutput1P8 1.8V output.
kDCDC_VoltageOutput1P0 1.0V output.

6.5 Function Documentation

6.5.1 void DCDC_Init (DCDC_Type * *base*, const dcdc_config_t * *config*)

Parameters

<i>base</i>	DCDC peripheral base address.
<i>config</i>	Pointer to the dcdc_config_t structure.

6.5.2 void DCDC_Deinit (DCDC_Type * *base*)

Parameters

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

6.5.3 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 dcdc_config_t .
---------------	--

6.5.4 static void DCDC_EnterLowPowerModeViaStandbyRequest (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"> • true Makes DCDC enter into low power mode for GPC standby mode.

6.5.5 static void DCDC_EnterLowPowerMode (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"> • true Makes DCDC enter into low power mode.

6.5.6 static void DCDC_EnterStandbyMode (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"> • true Makes DCDC enter into standby mode.

6.5.7 `static void DCDC_SetVDD1P0StandbyModeTargetVoltage (DCDC_Type *
base, dcdc_standby_mode_1P0_target_vol_t targetVoltage) [inline],
[static]`

Parameters

<i>base</i>	DCDC peripheral base address.
<i>targetVoltage</i>	The target value of VDD1P0 in standby mode, see dcdc_standby_mode_1P0_target_vol_t .

6.5.8 static uint16_t DCDC_GetVDD1P0StandbyModeTargetVoltage (DCDC_Type * *base*) [inline], [static]

Parameters

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

Returns

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

6.5.9 static void DCDC_SetVDD1P8StandbyModeTargetVoltage (DCDC_Type * *base*, dcdc_standby_mode_1P8_target_vol_t *targetVoltage*) [inline], [static]

Parameters

<i>base</i>	DCDC peripheral base address.
<i>targetVoltage</i>	The target value of VDD1P8 in standby mode, see dcdc_standby_mode_1P8_target_vol_t .

6.5.10 static uint16_t DCDC_GetVDD1P8StandbyModeTargetVoltage (DCDC_Type * *base*) [inline], [static]

Parameters

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

Returns

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

6.5.11 `static void DCDC_SetVDD1P0BuckModeTargetVoltage (DCDC_Type * base,
dcdc_buck_mode_1P0_target_vol_t targetVoltage) [inline], [static]`

Parameters

<i>base</i>	DCDC peripheral base address.
<i>targetVoltage</i>	The target value of VDD1P0 in buck mode, see dcdc_buck_mode_1P0_target_vol_t .

6.5.12 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".

6.5.13 static void DCDC_SetVDD1P8BuckModeTargetVoltage (DCDC_Type * *base*, dcdc_buck_mode_1P8_target_vol_t *targetVoltage*) [inline], [static]

Parameters

<i>base</i>	DCDC peripheral base address.
<i>targetVoltage</i>	The target value of VDD1P8 in buck mode, see dcdc_buck_mode_1P8_target_vol_t .

6.5.14 static uint16_t DCDC_GetVDD1P8BuckModeTargetVoltage (DCDC_Type * *base*) [inline], [static]

Parameters

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

Returns

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

6.5.15 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"> • true Enables stepping for VDD1P0. • false Disables stepping for VDD1P0.

6.5.16 static void DCDC_EnableVDD1P8TargetVoltageStepping (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"> • true Enables stepping for VDD1P8. • false Disables stepping for VDD1P8.

6.5.17 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 dcdc_detection_config_t .
---------------	--

6.5.18 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 dcdc_detection_config_t .

6.5.19 static void DCDC_EnableOutputRangeComparator (DCDC_Type * *base*, bool *enable*) [inline], [static]

The output range comparator is disabled by default.

Parameters

<i>base</i>	DCDC peripheral base address.
<i>enable</i>	Enable the feature or not. <ul style="list-style-type: none"> • true Enable the output range comparator. • false Disable the output range comparator.

6.5.20 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 dcdc_clock_source_t .

6.5.21 void DCDC_GetDefaultLowPowerConfig (dcdc_low_power_config_t * *config*)

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

```
* config->enableAdjustHystereticValue = false;
*
```


Parameters

<i>config</i>	Pointer to configuration structure. See to dcdc_low_power_config_t .
---------------	--

6.5.22 void DCDC_SetLowPowerConfig (DCDC_Type * *base*, const dcdc_low_power_config_t * *config*)

Parameters

<i>base</i>	DCDC peripheral base address.
<i>config</i>	Pointer to configuration structure. See to dcdc_low_power_config_t .

6.5.23 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.

6.5.24 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 dcdc_loop_control_config_t .
---------------	---

6.5.25 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 dcdc_loop_control_config_t .

6.5.26 void DCDC_SetMinPowerConfig (DCDC_Type * *base*, const *dcdc_min_power_config_t* * *config*)

Parameters

<i>base</i>	DCDC peripheral base address.
<i>config</i>	Pointer to configuration structure. See to dcdc_min_power_config_t .

6.5.27 static void DCDC_SetLPComparatorBiasValue (DCDC_Type * *base*, *dcdc_comparator_current_bias_t* *biasValue*) [inline], [static]

Parameters

<i>base</i>	DCDC peripheral base address.
<i>biasValue</i>	The current bias of low power comparator. Refer to dcdc_comparator_current_bias_t .

6.5.28 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 dcdc_internal_regulator_config_t .

6.5.29 `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.

6.5.30 static void DCDC_EnableImproveTransition (DCDC_Type * *base*, bool *enable*) [inline], [static]

Note

It is valid while zero cross detection is enabled. If output exceeds the threshold, DCDC would return CCM from DCM.

Parameters

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

6.5.31 void DCDC_SetPointInit (DCDC_Type * *base*, const dcdc_setpoint_config_t * *config*)

Note

The function should be invoked in the initial step to config the DCDC via setpoint control mode.

Parameters

<i>base</i>	DCDC peripheral base address.
<i>config</i>	The pointer to the structure dcdc_setpoint_config_t .

6.5.32 static void DCDC_SetPointDeinit (DCDC_Type * *base*, uint32_t *setpointMap*) [inline], [static]

Parameters

<i>base</i>	DCDC peripheral base address.
<i>setpointMap</i>	The map of the setpoint to disable the DCDC module, Should be the OR'ed value of <code>_dcdc_setpoint_map</code> .

6.5.33 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](#).

6.5.34 void DCDC_BootIntoDCM (DCDC_Type * *base*)

```
* pwd_zcd=0x0;
* DM_CTRL = 1'b1;
* pwd_cmp_offset=0x0;
* dcdc_loopctrl_en_rcscale=0x3 or 0x5;
* DCM_set_ctrl=1'b1;
*
```

Parameters

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

6.5.35 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 7

ROMAPI Driver

7.1 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, 1U)`)
ROM API version 1.1.1.
- #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.

7.2 Data Structure Documentation

7.2.1 struct serial_nor_config_option_t

7.2.2 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.

7.2.3 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.

7.2.4 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] Data setup time, default value: 3
- uint8_t [columnAddressWidth](#)
[0x00f-0x00f] Column Address with, 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 deviceModeTpe
- `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

7.2.5 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 ipcmdSerialClkFreq`
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`

- *Need to Restore NoCmd mode after IP command execution.*
- uint32_t [blockSize](#)
Block size.
- uint32_t [reserve2](#) [11]
Reserved for future use.

7.2.6 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.

7.3 Macro Definition Documentation

7.3.1 **#define FSL_ROM_ROMAPI_VERSION (MAKE_VERSION(1U, 1U, 1U))**

7.3.2 **#define FSL_ROM_FLEXSPINOR_DRIVER_VERSION (MAKE_VERSION(1U, 7U, 0U))**

7.3.3 **#define FSL_ROM_HAS_FLEXSPINOR_API (1)**

7.3.4 **#define FSL_ROM_HAS_RUNBOOTLOADER_API (1)**

7.3.5 **#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_GET_CONFIG (1)**

7.3.6 **#define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_FLASH_INIT (1)**

7.3.7 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE (1)

7.3.8 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_SECTOR (1)

7.3.9 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_BLOCK (1)

7.3.10 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_ERASE_ALL (1)

7.3.11 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_READ (1)

7.3.12 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_UPDATE_LUT (1)

7.3.13 #define FSL_ROM_FLEXSPINOR_API_HAS_FEATURE_CMD_XFER (1)

7.3.14 #define kROM_StatusGroup_FLEXSPINOR 201U

7.3.15 #define FLEXSPI_CFG_BLK_TAG (0x42464346UL)

ascii "FCFB" Big Endian

7.3.16 #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

7.4 Enumeration Type Documentation

7.4.1 anonymous enum

Enumerator

kSerialFlash_1Pad 1-wire communication
kSerialFlash_2Pads 2-wire communication
kSerialFlash_4Pads 4-wire communication
kSerialFlash_8Pads 8-wire communication

7.4.2 anonymous enum

Enumerator

kFLEXSPIClk_SDR Clock configure for SDR mode.

kFLEXSPIClk_DDR Clock configure for DDR mode.

7.4.3 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.

7.4.4 anonymous enum

Enumerator

kFLEXSPIDeviceType_SerialNOR Flash device is Serial NOR.

7.4.5 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.

7.4.6 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.

7.4.7 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.

7.4.8 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.

7.4.9 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.

7.4.10 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.

7.5 Function Documentation

7.5.1 void ROM_API_Init (void)

Get the bootloader api entry address.

7.5.2 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.
------------	---

7.5.3 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

7.5.4 **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

7.5.5 **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

7.5.6 **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

7.5.7 **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-SequenceExecution-Timeout</i>	Sequence Execution timeout.
<i>kStatus_ROM_FLEXSPI-DeviceTimeout</i>	the device timeout

7.5.8 **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

7.5.9 **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

7.5.10 **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_SequenceExecution-Timeout</i>	Sequence Execution timeout.
<i>kStatus_ROM_FLEXSPI_DeviceTimeout</i>	the device timeout

7.5.11 **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_SequenceExecution-Timeout</i>	Sequence Execution timeout.

7.5.12 **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_SequenceExecution-Timeout</i>	Sequence Execution timeout.

7.5.13 **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- _SequenceExecution- Timeout</i>	Sequence Execution timeout.
<i>kStatus_ROM_FLEXSPI- _InvalidSequence</i>	A invalid Sequence is provided.
<i>kStatus_ROM_FLEXSPI- _DeviceTimeout</i>	Device timeout.

7.5.14 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 8

IOMUXC: IOMUX Controller

8.1 Overview

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

Files

- file [fsl_iomuxc.h](#)

Driver version

- #define [FSL_IOMUXC_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 2))
IOMUXC driver version 2.0.2.

Pin function ID

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

- #define **IOMUXC_GPIO_LPSR_00_FLEXCAN3_TX** 0x40C08000U, 0x0U, 0, 0, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_MIC_CLK** 0x40C08000U, 0x1U, 0, 0, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_MQS_RIGHT** 0x40C08000U, 0x2U, 0, 0, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_ARM_CM4_EVENTO** 0x40C08000U, 0x3U, 0, 0, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_GPIO_MUX6_IO00** 0x40C08000U, 0x5U, 0, 0, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_LPUART12_TXD** 0x40C08000U, 0x6U, 0x40C080B0U, 0x0U, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_SAI4_MCLK** 0x40C08000U, 0x7U, 0x40C080C8U, 0x0U, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_00_GPIO12_IO00** 0x40C08000U, 0xAU, 0, 0, 0x40C08040U
- #define **IOMUXC_GPIO_LPSR_01_FLEXCAN3_RX** 0x40C08004U, 0x0U, 0x40C08080U, 0x0U, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_01_MIC_BITSTREAM0** 0x40C08004U, 0x1U, 0x40C080B4U, 0x0U, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_01_MQS_LEFT** 0x40C08004U, 0x2U, 0, 0, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_01_ARM_CM4_EVENTI** 0x40C08004U, 0x3U, 0, 0, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_01_GPIO_MUX6_IO01** 0x40C08004U, 0x5U, 0, 0, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_01_LPUART12_RXD** 0x40C08004U, 0x6U, 0x40C080ACU, 0x0U, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_01_GPIO12_IO01** 0x40C08004U, 0xAU, 0, 0, 0x40C08044U
- #define **IOMUXC_GPIO_LPSR_02_GPIO12_IO02** 0x40C08008U, 0xAU, 0, 0, 0x40C08048U

- #define **IOMUXC_GPIO_LPSR_02_SRC_BOOT_MODE00** 0x40C08008U, 0x0U, 0, 0, 0x40C08048U
- #define **IOMUXC_GPIO_LPSR_02_LPSPi5_SCK** 0x40C08008U, 0x1U, 0x40C08098U, 0x0U, 0x40C08048U
- #define **IOMUXC_GPIO_LPSR_02_SAI4_TX_DATA** 0x40C08008U, 0x2U, 0, 0, 0x40C08048U
- #define **IOMUXC_GPIO_LPSR_02_MQS_RIGHT** 0x40C08008U, 0x3U, 0, 0, 0x40C08048U
- #define **IOMUXC_GPIO_LPSR_02_GPIO_MUX6_IO02** 0x40C08008U, 0x5U, 0, 0, 0x40C08048U
- #define **IOMUXC_GPIO_LPSR_03_SRC_BOOT_MODE01** 0x40C0800CU, 0x0U, 0, 0, 0x40C0804CU
- #define **IOMUXC_GPIO_LPSR_03_LPSPi5_PCS0** 0x40C0800CU, 0x1U, 0x40C08094U, 0x0U, 0x40C0804CU
- #define **IOMUXC_GPIO_LPSR_03_SAI4_TX_SYNC** 0x40C0800CU, 0x2U, 0x40C080DCU, 0x0U, 0x40C0804CU
- #define **IOMUXC_GPIO_LPSR_03_MQS_LEFT** 0x40C0800CU, 0x3U, 0, 0, 0x40C0804CU
- #define **IOMUXC_GPIO_LPSR_03_GPIO_MUX6_IO03** 0x40C0800CU, 0x5U, 0, 0, 0x40C0804CU
- #define **IOMUXC_GPIO_LPSR_03_GPIO12_IO03** 0x40C0800CU, 0xAU, 0, 0, 0x40C0804CU
- #define **IOMUXC_GPIO_LPSR_04_LPI2C5_SDA** 0x40C08010U, 0x0U, 0x40C08088U, 0x0U, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_04_LPSPi5_SOUT** 0x40C08010U, 0x1U, 0x40C080A0U, 0x0U, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_04_SAI4_TX_BCLK** 0x40C08010U, 0x2U, 0x40C080D8U, 0x0U, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_04_LPUART12_RTS_B** 0x40C08010U, 0x3U, 0, 0, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_04_GPIO_MUX6_IO04** 0x40C08010U, 0x5U, 0, 0, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_04_LPUART11_TXD** 0x40C08010U, 0x6U, 0x40C080A8U, 0x0U, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_04_GPIO12_IO04** 0x40C08010U, 0xAU, 0, 0, 0x40C08050U
- #define **IOMUXC_GPIO_LPSR_05_GPIO12_IO05** 0x40C08014U, 0xAU, 0, 0, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_LPI2C5_SCL** 0x40C08014U, 0x0U, 0x40C08084U, 0x0U, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_LPSPi5_SIN** 0x40C08014U, 0x1U, 0x40C0809CU, 0x0U, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_SAI4_MCLK** 0x40C08014U, 0x2U, 0x40C080C8U, 0x1U, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_LPUART12_CTS_B** 0x40C08014U, 0x3U, 0, 0, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_GPIO_MUX6_IO05** 0x40C08014U, 0x5U, 0, 0, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_LPUART11_RXD** 0x40C08014U, 0x6U, 0x40C080A4U, 0x0U, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_05_NMI_GLUE_NMI** 0x40C08014U, 0x7U, 0x40C080C4U, 0x0U, 0x40C08054U
- #define **IOMUXC_GPIO_LPSR_06_LPI2C6_SDA** 0x40C08018U, 0x0U, 0x40C08090U, 0x0U, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_SAI4_RX_DATA** 0x40C08018U, 0x2U, 0x40C080D0U,

- 0x0U, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_LPUART12_TXD** 0x40C08018U, 0x3U, 0x40C080B0U, 0x1U, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_LPSPi6_PCS3** 0x40C08018U, 0x4U, 0, 0, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_GPIO_MUX6_IO06** 0x40C08018U, 0x5U, 0, 0, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_FLEXCAN3_TX** 0x40C08018U, 0x6U, 0, 0, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_PIT2_TRIGGER3** 0x40C08018U, 0x7U, 0, 0, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_LPSPi5_PCS1** 0x40C08018U, 0x8U, 0, 0, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_06_GPIO12_IO06** 0x40C08018U, 0xAU, 0, 0, 0x40C08058U
- #define **IOMUXC_GPIO_LPSR_07_LPI2C6_SCL** 0x40C0801CU, 0x0U, 0x40C0808CU, 0x0U, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_SAI4_RX_BCLK** 0x40C0801CU, 0x2U, 0x40C080CCU, 0x0U, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_LPUART12_RXD** 0x40C0801CU, 0x3U, 0x40C080ACU, 0x1U, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_LPSPi6_PCS2** 0x40C0801CU, 0x4U, 0, 0, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_GPIO_MUX6_IO07** 0x40C0801CU, 0x5U, 0, 0, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_FLEXCAN3_RX** 0x40C0801CU, 0x6U, 0x40C08080U, 0x1U, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_PIT2_TRIGGER2** 0x40C0801CU, 0x7U, 0, 0, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_LPSPi5_PCS2** 0x40C0801CU, 0x8U, 0, 0, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_07_GPIO12_IO07** 0x40C0801CU, 0xAU, 0, 0, 0x40C0805CU
- #define **IOMUXC_GPIO_LPSR_08_GPIO12_IO08** 0x40C08020U, 0xAU, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_LPUART11_TXD** 0x40C08020U, 0x0U, 0x40C080A8U, 0x1U, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_FLEXCAN3_TX** 0x40C08020U, 0x1U, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_SAI4_RX_SYNC** 0x40C08020U, 0x2U, 0x40C080D4U, 0x0U, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_MIC_CLK** 0x40C08020U, 0x3U, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_LPSPi6_PCS1** 0x40C08020U, 0x4U, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_GPIO_MUX6_IO08** 0x40C08020U, 0x5U, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_LPI2C5_SDA** 0x40C08020U, 0x6U, 0x40C08088U, 0x1U, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_PIT2_TRIGGER1** 0x40C08020U, 0x7U, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_08_LPSPi5_PCS3** 0x40C08020U, 0x8U, 0, 0, 0x40C08060U
- #define **IOMUXC_GPIO_LPSR_09_GPIO12_IO09** 0x40C08024U, 0xAU, 0, 0, 0x40C08064U
- #define **IOMUXC_GPIO_LPSR_09_LPUART11_RXD** 0x40C08024U, 0x0U, 0x40C080A4U, 0x1U, 0x40C08064U
- #define **IOMUXC_GPIO_LPSR_09_FLEXCAN3_RX** 0x40C08024U, 0x1U, 0x40C08080U, 0x2U, 0x40C08064U
- #define **IOMUXC_GPIO_LPSR_09_PIT2_TRIGGER0** 0x40C08024U, 0x2U, 0, 0, 0x40C08064U
- #define **IOMUXC_GPIO_LPSR_09_MIC_BITSTREAM0** 0x40C08024U, 0x3U, 0x40C080B4U

```

U, 0x1U, 0x40C08064U
• #define IOMUXC_GPIO_LPSR_09_LPSPi6_PCS0 0x40C08024U, 0x4U, 0, 0, 0x40C08064U
• #define IOMUXC_GPIO_LPSR_09_GPIO_MUX6_IO09 0x40C08024U, 0x5U, 0, 0, 0x40-
  C08064U
• #define IOMUXC_GPIO_LPSR_09_LPI2C5_SCL 0x40C08024U, 0x6U, 0x40C08084U, 0x1U,
  0x40C08064U
• #define IOMUXC_GPIO_LPSR_09_SAI4_TX_DATA 0x40C08024U, 0x7U, 0, 0, 0x40C08064-
  U
• #define IOMUXC_GPIO_LPSR_10_GPIO12_IO10 0x40C08028U, 0xAU, 0, 0, 0x40C08068U
• #define IOMUXC_GPIO_LPSR_10_JTAG_MUX_TRSTB 0x40C08028U, 0x0U, 0, 0, 0x40-
  C08068U
• #define IOMUXC_GPIO_LPSR_10_LPUART11_CTS_B 0x40C08028U, 0x1U, 0, 0, 0x40-
  C08068U
• #define IOMUXC_GPIO_LPSR_10_LPI2C6_SDA 0x40C08028U, 0x2U, 0x40C08090U, 0x1U,
  0x40C08068U
• #define IOMUXC_GPIO_LPSR_10_MIC_BITSTREAM1 0x40C08028U, 0x3U, 0x40C080B8-
  U, 0x0U, 0x40C08068U
• #define IOMUXC_GPIO_LPSR_10_LPSPi6_SCK 0x40C08028U, 0x4U, 0, 0, 0x40C08068U
• #define IOMUXC_GPIO_LPSR_10_GPIO_MUX6_IO10 0x40C08028U, 0x5U, 0, 0, 0x40-
  C08068U
• #define IOMUXC_GPIO_LPSR_10_LPI2C5_SCLS 0x40C08028U, 0x6U, 0, 0, 0x40C08068U
• #define IOMUXC_GPIO_LPSR_10_SAI4_TX_SYNC 0x40C08028U, 0x7U, 0x40C080DCU,
  0x1U, 0x40C08068U
• #define IOMUXC_GPIO_LPSR_10_LPUART12_TXD 0x40C08028U, 0x8U, 0x40C080B0U,
  0x2U, 0x40C08068U
• #define IOMUXC_GPIO_LPSR_11_JTAG_MUX_TDO 0x40C0802CU, 0x0U, 0, 0, 0x40-
  C0806CU
• #define IOMUXC_GPIO_LPSR_11_LPUART11_RTS_B 0x40C0802CU, 0x1U, 0, 0, 0x40-
  C0806CU
• #define IOMUXC_GPIO_LPSR_11_LPI2C6_SCL 0x40C0802CU, 0x2U, 0x40C0808CU, 0x1U,
  0x40C0806CU
• #define IOMUXC_GPIO_LPSR_11_MIC_BITSTREAM2 0x40C0802CU, 0x3U, 0x40C080B-
  CU, 0x0U, 0x40C0806CU
• #define IOMUXC_GPIO_LPSR_11_LPSPi6_SOUT 0x40C0802CU, 0x4U, 0, 0, 0x40C0806CU
• #define IOMUXC_GPIO_LPSR_11_GPIO_MUX6_IO11 0x40C0802CU, 0x5U, 0, 0, 0x40-
  C0806CU
• #define IOMUXC_GPIO_LPSR_11_LPI2C5_SDAS 0x40C0802CU, 0x6U, 0, 0, 0x40C0806CU
• #define IOMUXC_GPIO_LPSR_11_ARM_TRACE_SWO 0x40C0802CU, 0x7U, 0, 0, 0x40-
  C0806CU
• #define IOMUXC_GPIO_LPSR_11_LPUART12_RXD 0x40C0802CU, 0x8U, 0x40C080ACU,
  0x2U, 0x40C0806CU
• #define IOMUXC_GPIO_LPSR_11_GPIO12_IO11 0x40C0802CU, 0xAU, 0, 0, 0x40C0806CU
• #define IOMUXC_GPIO_LPSR_12_GPIO12_IO12 0x40C08030U, 0xAU, 0, 0, 0x40C08070U
• #define IOMUXC_GPIO_LPSR_12_JTAG_MUX_TDI 0x40C08030U, 0x0U, 0, 0, 0x40-
  C08070U
• #define IOMUXC_GPIO_LPSR_12_PIT2_TRIGGER0 0x40C08030U, 0x1U, 0, 0, 0x40-
  C08070U
• #define IOMUXC_GPIO_LPSR_12_MIC_BITSTREAM3 0x40C08030U, 0x3U, 0x40C080C0-
  U, 0x0U, 0x40C08070U
• #define IOMUXC_GPIO_LPSR_12_LPSPi6_SIN 0x40C08030U, 0x4U, 0, 0, 0x40C08070U
• #define IOMUXC_GPIO_LPSR_12_GPIO_MUX6_IO12 0x40C08030U, 0x5U, 0, 0, 0x40-

```

```

C08070U
• #define IOMUXC_GPIO_LPSR_12_LPI2C5_HREQ 0x40C08030U, 0x6U, 0, 0, 0x40C08070U
• #define IOMUXC_GPIO_LPSR_12_SAI4_TX_BCLK 0x40C08030U, 0x7U, 0x40C080D8U,
  0x1U, 0x40C08070U
• #define IOMUXC_GPIO_LPSR_12_LPSPI5_SCK 0x40C08030U, 0x8U, 0x40C08098U, 0x1U,
  0x40C08070U
• #define IOMUXC_GPIO_LPSR_13_GPIO12_IO13 0x40C08034U, 0xAU, 0, 0, 0x40C08074U
• #define IOMUXC_GPIO_LPSR_13_JTAG_MUX_MOD 0x40C08034U, 0x0U, 0, 0, 0x40-
  C08074U
• #define IOMUXC_GPIO_LPSR_13_MIC_BITSTREAM1 0x40C08034U, 0x1U, 0x40C080B8-
  U, 0x1U, 0x40C08074U
• #define IOMUXC_GPIO_LPSR_13_PIT2_TRIGGER1 0x40C08034U, 0x2U, 0, 0, 0x40-
  C08074U
• #define IOMUXC_GPIO_LPSR_13_GPIO_MUX6_IO13 0x40C08034U, 0x5U, 0, 0, 0x40-
  C08074U
• #define IOMUXC_GPIO_LPSR_13_SAI4_RX_DATA 0x40C08034U, 0x7U, 0x40C080D0U,
  0x1U, 0x40C08074U
• #define IOMUXC_GPIO_LPSR_13_LPSPI5_PCS0 0x40C08034U, 0x8U, 0x40C08094U, 0x1-
  U, 0x40C08074U
• #define IOMUXC_GPIO_LPSR_14_JTAG_MUX_TCK 0x40C08038U, 0x0U, 0, 0, 0x40-
  C08078U
• #define IOMUXC_GPIO_LPSR_14_MIC_BITSTREAM2 0x40C08038U, 0x1U, 0x40C080BC-
  U, 0x1U, 0x40C08078U
• #define IOMUXC_GPIO_LPSR_14_PIT2_TRIGGER2 0x40C08038U, 0x2U, 0, 0, 0x40-
  C08078U
• #define IOMUXC_GPIO_LPSR_14_GPIO_MUX6_IO14 0x40C08038U, 0x5U, 0, 0, 0x40-
  C08078U
• #define IOMUXC_GPIO_LPSR_14_SAI4_RX_BCLK 0x40C08038U, 0x7U, 0x40C080CCU,
  0x1U, 0x40C08078U
• #define IOMUXC_GPIO_LPSR_14_LPSPI5_SOUT 0x40C08038U, 0x8U, 0x40C080A0U, 0x1-
  U, 0x40C08078U
• #define IOMUXC_GPIO_LPSR_14_GPIO12_IO14 0x40C08038U, 0xAU, 0, 0, 0x40C08078U
• #define IOMUXC_GPIO_LPSR_15_GPIO12_IO15 0x40C0803CU, 0xAU, 0, 0, 0x40C0807CU
• #define IOMUXC_GPIO_LPSR_15_JTAG_MUX_TMS 0x40C0803CU, 0x0U, 0, 0, 0x40-
  C0807CU
• #define IOMUXC_GPIO_LPSR_15_MIC_BITSTREAM3 0x40C0803CU, 0x1U, 0x40C080C0-
  U, 0x1U, 0x40C0807CU
• #define IOMUXC_GPIO_LPSR_15_PIT2_TRIGGER3 0x40C0803CU, 0x2U, 0, 0, 0x40C0807-
  CU
• #define IOMUXC_GPIO_LPSR_15_GPIO_MUX6_IO15 0x40C0803CU, 0x5U, 0, 0, 0x40-
  C0807CU
• #define IOMUXC_GPIO_LPSR_15_SAI4_RX_SYNC 0x40C0803CU, 0x7U, 0x40C080D4U,
  0x1U, 0x40C0807CU
• #define IOMUXC_GPIO_LPSR_15_LPSPI5_SIN 0x40C0803CU, 0x8U, 0x40C0809CU, 0x1U,
  0x40C0807CU
• #define IOMUXC_WAKEUP_DIG_GPIO13_IO00 0x40C94000U, 0x5U, 0, 0, 0x40C94040U
• #define IOMUXC_WAKEUP_DIG_NMI_GLUE_NMI 0x40C94000U, 0x7U, 0x40C080C4U,
  0x1U, 0x40C94040U
• #define IOMUXC_PMIC_ON_REQ_DIG_SNVS_LP_PMIC_ON_REQ 0x40C94004U, 0x0U,
  0, 0, 0x40C94044U

```


- #define **IOMUXC_PMIC_ON_REQ_DIG_GPIO13_IO01** 0x40C94004U, 0x5U, 0, 0, 0x40-C94044U
- #define **IOMUXC_PMIC_STBY_REQ_DIG_CCM_PMIC_VSTBY_REQ** 0x40C94008U, 0x0-U, 0, 0, 0x40C94048U
- #define **IOMUXC_PMIC_STBY_REQ_DIG_GPIO13_IO02** 0x40C94008U, 0x5U, 0, 0, 0x40-C94048U
- #define **IOMUXC_GPIO_SNVS_00_DIG_SNVS_TAMPER0** 0x40C9400CU, 0x0U, 0, 0, 0x40-C9404CU
- #define **IOMUXC_GPIO_SNVS_00_DIG_GPIO13_IO03** 0x40C9400CU, 0x5U, 0, 0, 0x40-C9404CU
- #define **IOMUXC_GPIO_SNVS_01_DIG_SNVS_TAMPER1** 0x40C94010U, 0x0U, 0, 0, 0x40-C94050U
- #define **IOMUXC_GPIO_SNVS_01_DIG_GPIO13_IO04** 0x40C94010U, 0x5U, 0, 0, 0x40-C94050U
- #define **IOMUXC_GPIO_SNVS_02_DIG_SNVS_TAMPER2** 0x40C94014U, 0x0U, 0, 0, 0x40-C94054U
- #define **IOMUXC_GPIO_SNVS_02_DIG_GPIO13_IO05** 0x40C94014U, 0x5U, 0, 0, 0x40-C94054U
- #define **IOMUXC_GPIO_SNVS_03_DIG_SNVS_TAMPER3** 0x40C94018U, 0x0U, 0, 0, 0x40-C94058U
- #define **IOMUXC_GPIO_SNVS_03_DIG_GPIO13_IO06** 0x40C94018U, 0x5U, 0, 0, 0x40-C94058U
- #define **IOMUXC_GPIO_SNVS_04_DIG_SNVS_TAMPER4** 0x40C9401CU, 0x0U, 0, 0, 0x40-C9405CU
- #define **IOMUXC_GPIO_SNVS_04_DIG_GPIO13_IO07** 0x40C9401CU, 0x5U, 0, 0, 0x40-C9405CU
- #define **IOMUXC_GPIO_SNVS_05_DIG_SNVS_TAMPER5** 0x40C94020U, 0x0U, 0, 0, 0x40-C94060U
- #define **IOMUXC_GPIO_SNVS_05_DIG_GPIO13_IO08** 0x40C94020U, 0x5U, 0, 0, 0x40-C94060U
- #define **IOMUXC_GPIO_SNVS_06_DIG_SNVS_TAMPER6** 0x40C94024U, 0x0U, 0, 0, 0x40-C94064U
- #define **IOMUXC_GPIO_SNVS_06_DIG_GPIO13_IO09** 0x40C94024U, 0x5U, 0, 0, 0x40-C94064U
- #define **IOMUXC_GPIO_SNVS_07_DIG_SNVS_TAMPER7** 0x40C94028U, 0x0U, 0, 0, 0x40-C94068U
- #define **IOMUXC_GPIO_SNVS_07_DIG_GPIO13_IO10** 0x40C94028U, 0x5U, 0, 0, 0x40-C94068U
- #define **IOMUXC_GPIO_SNVS_08_DIG_SNVS_TAMPER8** 0x40C9402CU, 0x0U, 0, 0, 0x40-C9406CU
- #define **IOMUXC_GPIO_SNVS_08_DIG_GPIO13_IO11** 0x40C9402CU, 0x5U, 0, 0, 0x40-C9406CU
- #define **IOMUXC_GPIO_SNVS_09_DIG_SNVS_TAMPER9** 0x40C94030U, 0x0U, 0, 0, 0x40-C94070U
- #define **IOMUXC_GPIO_SNVS_09_DIG_GPIO13_IO12** 0x40C94030U, 0x5U, 0, 0, 0x40-C94070U
- #define **IOMUXC_TEST_MODE_DIG** 0, 0, 0, 0, 0x40C94034U
- #define **IOMUXC_POR_B_DIG** 0, 0, 0, 0, 0x40C94038U
- #define **IOMUXC_ONOFF_DIG** 0, 0, 0, 0, 0x40C9403CU
- #define **IOMUXC_GPIO_EMC_B1_00_SEMC_DATA00** 0x400E8010U, 0x0U, 0, 0, 0x400-

- E8254U
- #define **IOMUXC_GPIO_EMC_B1_00_FLEXPWM4_PWM0_A** 0x400E8010U, 0x1U, 0, 0, 0x400E8254U
- #define **IOMUXC_GPIO_EMC_B1_00_GPIO_MUX1_IO00** 0x400E8010U, 0x5U, 0, 0, 0x400E8254U
- #define **IOMUXC_GPIO_EMC_B1_00_FLEXIO1_D00** 0x400E8010U, 0x8U, 0, 0, 0x400E8254U
- #define **IOMUXC_GPIO_EMC_B1_00_GPIO7_IO00** 0x400E8010U, 0xAU, 0, 0, 0x400E8254U
- #define **IOMUXC_GPIO_EMC_B1_01_GPIO7_IO01** 0x400E8014U, 0xAU, 0, 0, 0x400E8258U
- #define **IOMUXC_GPIO_EMC_B1_01_SEMC_DATA01** 0x400E8014U, 0x0U, 0, 0, 0x400E8258U
- #define **IOMUXC_GPIO_EMC_B1_01_FLEXPWM4_PWM0_B** 0x400E8014U, 0x1U, 0, 0, 0x400E8258U
- #define **IOMUXC_GPIO_EMC_B1_01_GPIO_MUX1_IO01** 0x400E8014U, 0x5U, 0, 0, 0x400E8258U
- #define **IOMUXC_GPIO_EMC_B1_01_FLEXIO1_D01** 0x400E8014U, 0x8U, 0, 0, 0x400E8258U
- #define **IOMUXC_GPIO_EMC_B1_02_SEMC_DATA02** 0x400E8018U, 0x0U, 0, 0, 0x400E825CU
- #define **IOMUXC_GPIO_EMC_B1_02_FLEXPWM4_PWM1_A** 0x400E8018U, 0x1U, 0, 0, 0x400E825CU
- #define **IOMUXC_GPIO_EMC_B1_02_GPIO_MUX1_IO02** 0x400E8018U, 0x5U, 0, 0, 0x400E825CU
- #define **IOMUXC_GPIO_EMC_B1_02_FLEXIO1_D02** 0x400E8018U, 0x8U, 0, 0, 0x400E825CU
- #define **IOMUXC_GPIO_EMC_B1_02_GPIO7_IO02** 0x400E8018U, 0xAU, 0, 0, 0x400E825CU
- #define **IOMUXC_GPIO_EMC_B1_03_SEMC_DATA03** 0x400E801CU, 0x0U, 0, 0, 0x400E8260U
- #define **IOMUXC_GPIO_EMC_B1_03_FLEXPWM4_PWM1_B** 0x400E801CU, 0x1U, 0, 0, 0x400E8260U
- #define **IOMUXC_GPIO_EMC_B1_03_GPIO_MUX1_IO03** 0x400E801CU, 0x5U, 0, 0, 0x400E8260U
- #define **IOMUXC_GPIO_EMC_B1_03_FLEXIO1_D03** 0x400E801CU, 0x8U, 0, 0, 0x400E8260U
- #define **IOMUXC_GPIO_EMC_B1_03_GPIO7_IO03** 0x400E801CU, 0xAU, 0, 0, 0x400E8260U
- #define **IOMUXC_GPIO_EMC_B1_04_GPIO7_IO04** 0x400E8020U, 0xAU, 0, 0, 0x400E8264U
- #define **IOMUXC_GPIO_EMC_B1_04_SEMC_DATA04** 0x400E8020U, 0x0U, 0, 0, 0x400E8264U
- #define **IOMUXC_GPIO_EMC_B1_04_FLEXPWM4_PWM2_A** 0x400E8020U, 0x1U, 0, 0, 0x400E8264U
- #define **IOMUXC_GPIO_EMC_B1_04_GPIO_MUX1_IO04** 0x400E8020U, 0x5U, 0, 0, 0x400E8264U
- #define **IOMUXC_GPIO_EMC_B1_04_FLEXIO1_D04** 0x400E8020U, 0x8U, 0, 0, 0x400E8264U
- #define **IOMUXC_GPIO_EMC_B1_05_SEMC_DATA05** 0x400E8024U, 0x0U, 0, 0, 0x400E8268U

- E8268U
- #define **IOMUXC_GPIO_EMC_B1_05_FLEXPWM4_PWM2_B** 0x400E8024U, 0x1U, 0, 0, 0x400E8268U
- #define **IOMUXC_GPIO_EMC_B1_05_GPIO_MUX1_IO05** 0x400E8024U, 0x5U, 0, 0, 0x400E8268U
- #define **IOMUXC_GPIO_EMC_B1_05_FLEXIO1_D05** 0x400E8024U, 0x8U, 0, 0, 0x400E8268U
- #define **IOMUXC_GPIO_EMC_B1_05_GPIO7_IO05** 0x400E8024U, 0xAU, 0, 0, 0x400E8268U
- #define **IOMUXC_GPIO_EMC_B1_06_SEMC_DATA06** 0x400E8028U, 0x0U, 0, 0, 0x400E826CU
- #define **IOMUXC_GPIO_EMC_B1_06_FLEXPWM2_PWM0_A** 0x400E8028U, 0x1U, 0x400E8518U, 0x0U, 0x400E826CU
- #define **IOMUXC_GPIO_EMC_B1_06_GPIO_MUX1_IO06** 0x400E8028U, 0x5U, 0, 0, 0x400E826CU
- #define **IOMUXC_GPIO_EMC_B1_06_FLEXIO1_D06** 0x400E8028U, 0x8U, 0, 0, 0x400E826CU
- #define **IOMUXC_GPIO_EMC_B1_06_GPIO7_IO06** 0x400E8028U, 0xAU, 0, 0, 0x400E826CU
- #define **IOMUXC_GPIO_EMC_B1_07_GPIO7_IO07** 0x400E802CU, 0xAU, 0, 0, 0x400E8270U
- #define **IOMUXC_GPIO_EMC_B1_07_SEMC_DATA07** 0x400E802CU, 0x0U, 0, 0, 0x400E8270U
- #define **IOMUXC_GPIO_EMC_B1_07_FLEXPWM2_PWM0_B** 0x400E802CU, 0x1U, 0x400E8524U, 0x0U, 0x400E8270U
- #define **IOMUXC_GPIO_EMC_B1_07_GPIO_MUX1_IO07** 0x400E802CU, 0x5U, 0, 0, 0x400E8270U
- #define **IOMUXC_GPIO_EMC_B1_07_FLEXIO1_D07** 0x400E802CU, 0x8U, 0, 0, 0x400E8270U
- #define **IOMUXC_GPIO_EMC_B1_08_SEMC_DM00** 0x400E8030U, 0x0U, 0, 0, 0x400E8274U
- #define **IOMUXC_GPIO_EMC_B1_08_FLEXPWM2_PWM1_A** 0x400E8030U, 0x1U, 0x400E851CU, 0x0U, 0x400E8274U
- #define **IOMUXC_GPIO_EMC_B1_08_GPIO_MUX1_IO08** 0x400E8030U, 0x5U, 0, 0, 0x400E8274U
- #define **IOMUXC_GPIO_EMC_B1_08_FLEXIO1_D08** 0x400E8030U, 0x8U, 0, 0, 0x400E8274U
- #define **IOMUXC_GPIO_EMC_B1_08_GPIO7_IO08** 0x400E8030U, 0xAU, 0, 0, 0x400E8274U
- #define **IOMUXC_GPIO_EMC_B1_09_SEMC_ADDR00** 0x400E8034U, 0x0U, 0, 0, 0x400E8278U
- #define **IOMUXC_GPIO_EMC_B1_09_FLEXPWM2_PWM1_B** 0x400E8034U, 0x1U, 0x400E8528U, 0x0U, 0x400E8278U
- #define **IOMUXC_GPIO_EMC_B1_09_GPT5_CAPTURE1** 0x400E8034U, 0x2U, 0, 0, 0x400E8278U
- #define **IOMUXC_GPIO_EMC_B1_09_GPIO_MUX1_IO09** 0x400E8034U, 0x5U, 0, 0, 0x400E8278U
- #define **IOMUXC_GPIO_EMC_B1_09_FLEXIO1_D09** 0x400E8034U, 0x8U, 0, 0, 0x400E8278U
- #define **IOMUXC_GPIO_EMC_B1_09_GPIO7_IO09** 0x400E8034U, 0xAU, 0, 0, 0x400E8278U

- U
- #define **IOMUXC_GPIO_EMC_B1_10_SEMC_ADDR01** 0x400E8038U, 0x0U, 0, 0, 0x400-E827CU
 - #define **IOMUXC_GPIO_EMC_B1_10_FLEXPWM2_PWM2_A** 0x400E8038U, 0x1U, 0x400-E8520U, 0x0U, 0x400E827CU
 - #define **IOMUXC_GPIO_EMC_B1_10_GPT5_CAPTURE2** 0x400E8038U, 0x2U, 0, 0, 0x400-E827CU
 - #define **IOMUXC_GPIO_EMC_B1_10_GPIO_MUX1_IO10** 0x400E8038U, 0x5U, 0, 0, 0x400-E827CU
 - #define **IOMUXC_GPIO_EMC_B1_10_FLEXIO1_D10** 0x400E8038U, 0x8U, 0, 0, 0x400E827-CU
 - #define **IOMUXC_GPIO_EMC_B1_10_GPIO7_IO10** 0x400E8038U, 0xAU, 0, 0, 0x400E827-CU
 - #define **IOMUXC_GPIO_EMC_B1_11_GPIO7_IO11** 0x400E803CU, 0xAU, 0, 0, 0x400E8280-U
 - #define **IOMUXC_GPIO_EMC_B1_11_SEMC_ADDR02** 0x400E803CU, 0x0U, 0, 0, 0x400-E8280U
 - #define **IOMUXC_GPIO_EMC_B1_11_FLEXPWM2_PWM2_B** 0x400E803CU, 0x1U, 0x400-E852CU, 0x0U, 0x400E8280U
 - #define **IOMUXC_GPIO_EMC_B1_11_GPT5_COMPARE1** 0x400E803CU, 0x2U, 0, 0, 0x400E8280U
 - #define **IOMUXC_GPIO_EMC_B1_11_GPIO_MUX1_IO11** 0x400E803CU, 0x5U, 0, 0, 0x400-E8280U
 - #define **IOMUXC_GPIO_EMC_B1_11_FLEXIO1_D11** 0x400E803CU, 0x8U, 0, 0, 0x400-E8280U
 - #define **IOMUXC_GPIO_EMC_B1_12_SEMC_ADDR03** 0x400E8040U, 0x0U, 0, 0, 0x400-E8284U
 - #define **IOMUXC_GPIO_EMC_B1_12_XBAR1_INOUT04** 0x400E8040U, 0x1U, 0, 0, 0x400-E8284U
 - #define **IOMUXC_GPIO_EMC_B1_12_GPT5_COMPARE2** 0x400E8040U, 0x2U, 0, 0, 0x400-E8284U
 - #define **IOMUXC_GPIO_EMC_B1_12_GPIO_MUX1_IO12** 0x400E8040U, 0x5U, 0, 0, 0x400-E8284U
 - #define **IOMUXC_GPIO_EMC_B1_12_FLEXIO1_D12** 0x400E8040U, 0x8U, 0, 0, 0x400-E8284U
 - #define **IOMUXC_GPIO_EMC_B1_12_GPIO7_IO12** 0x400E8040U, 0xAU, 0, 0, 0x400E8284-U
 - #define **IOMUXC_GPIO_EMC_B1_13_SEMC_ADDR04** 0x400E8044U, 0x0U, 0, 0, 0x400-E8288U
 - #define **IOMUXC_GPIO_EMC_B1_13_XBAR1_INOUT05** 0x400E8044U, 0x1U, 0, 0, 0x400-E8288U
 - #define **IOMUXC_GPIO_EMC_B1_13_GPT5_COMPARE3** 0x400E8044U, 0x2U, 0, 0, 0x400-E8288U
 - #define **IOMUXC_GPIO_EMC_B1_13_GPIO_MUX1_IO13** 0x400E8044U, 0x5U, 0, 0, 0x400-E8288U
 - #define **IOMUXC_GPIO_EMC_B1_13_FLEXIO1_D13** 0x400E8044U, 0x8U, 0, 0, 0x400-E8288U
 - #define **IOMUXC_GPIO_EMC_B1_13_GPIO7_IO13** 0x400E8044U, 0xAU, 0, 0, 0x400E8288-U
 - #define **IOMUXC_GPIO_EMC_B1_14_GPIO7_IO14** 0x400E8048U, 0xAU, 0, 0, 0x400E828-

- CU
- #define **IOMUXC_GPIO_EMC_B1_14_SEMC_ADDR05** 0x400E8048U, 0x0U, 0, 0, 0x400-E828CU
- #define **IOMUXC_GPIO_EMC_B1_14_XBAR1_INOUT06** 0x400E8048U, 0x1U, 0, 0, 0x400-E828CU
- #define **IOMUXC_GPIO_EMC_B1_14_GPT5_CLK** 0x400E8048U, 0x2U, 0, 0, 0x400E828CU
- #define **IOMUXC_GPIO_EMC_B1_14_GPIO_MUX1_IO14** 0x400E8048U, 0x5U, 0, 0, 0x400-E828CU
- #define **IOMUXC_GPIO_EMC_B1_14_FLEXIO1_D14** 0x400E8048U, 0x8U, 0, 0, 0x400E828-CU
- #define **IOMUXC_GPIO_EMC_B1_15_SEMC_ADDR06** 0x400E804CU, 0x0U, 0, 0, 0x400-E8290U
- #define **IOMUXC_GPIO_EMC_B1_15_XBAR1_INOUT07** 0x400E804CU, 0x1U, 0, 0, 0x400-E8290U
- #define **IOMUXC_GPIO_EMC_B1_15_GPIO_MUX1_IO15** 0x400E804CU, 0x5U, 0, 0, 0x400-E8290U
- #define **IOMUXC_GPIO_EMC_B1_15_FLEXIO1_D15** 0x400E804CU, 0x8U, 0, 0, 0x400-E8290U
- #define **IOMUXC_GPIO_EMC_B1_15_GPIO7_IO15** 0x400E804CU, 0xAU, 0, 0, 0x400E8290-U
- #define **IOMUXC_GPIO_EMC_B1_16_SEMC_ADDR07** 0x400E8050U, 0x0U, 0, 0, 0x400-E8294U
- #define **IOMUXC_GPIO_EMC_B1_16_XBAR1_INOUT08** 0x400E8050U, 0x1U, 0, 0, 0x400-E8294U
- #define **IOMUXC_GPIO_EMC_B1_16_GPIO_MUX1_IO16** 0x400E8050U, 0x5U, 0, 0, 0x400-E8294U
- #define **IOMUXC_GPIO_EMC_B1_16_FLEXIO1_D16** 0x400E8050U, 0x8U, 0, 0, 0x400-E8294U
- #define **IOMUXC_GPIO_EMC_B1_16_GPIO7_IO16** 0x400E8050U, 0xAU, 0, 0, 0x400E8294-U
- #define **IOMUXC_GPIO_EMC_B1_17_GPIO7_IO17** 0x400E8054U, 0xAU, 0, 0, 0x400E8298-U
- #define **IOMUXC_GPIO_EMC_B1_17_SEMC_ADDR08** 0x400E8054U, 0x0U, 0, 0, 0x400-E8298U
- #define **IOMUXC_GPIO_EMC_B1_17_FLEXPWM4_PWM3_A** 0x400E8054U, 0x1U, 0, 0, 0x400E8298U
- #define **IOMUXC_GPIO_EMC_B1_17_TMR1_TIMER0** 0x400E8054U, 0x2U, 0x400E863CU, 0x0U, 0x400E8298U
- #define **IOMUXC_GPIO_EMC_B1_17_GPIO_MUX1_IO17** 0x400E8054U, 0x5U, 0, 0, 0x400-E8298U
- #define **IOMUXC_GPIO_EMC_B1_17_FLEXIO1_D17** 0x400E8054U, 0x8U, 0, 0, 0x400-E8298U
- #define **IOMUXC_GPIO_EMC_B1_18_SEMC_ADDR09** 0x400E8058U, 0x0U, 0, 0, 0x400-E829CU
- #define **IOMUXC_GPIO_EMC_B1_18_FLEXPWM4_PWM3_B** 0x400E8058U, 0x1U, 0, 0, 0x400E829CU
- #define **IOMUXC_GPIO_EMC_B1_18_TMR2_TIMER0** 0x400E8058U, 0x2U, 0x400E8648U, 0x0U, 0x400E829CU
- #define **IOMUXC_GPIO_EMC_B1_18_GPIO_MUX1_IO18** 0x400E8058U, 0x5U, 0, 0, 0x400-E829CU

- #define **IOMUXC_GPIO_EMC_B1_18_FLEXIO1_D18** 0x400E8058U, 0x8U, 0, 0, 0x400E829-CU
- #define **IOMUXC_GPIO_EMC_B1_18_GPIO7_IO18** 0x400E8058U, 0xAU, 0, 0, 0x400E829-CU
- #define **IOMUXC_GPIO_EMC_B1_19_SEMC_ADDR11** 0x400E805CU, 0x0U, 0, 0, 0x400-E82A0U
- #define **IOMUXC_GPIO_EMC_B1_19_FLEXPWM2_PWM3_A** 0x400E805CU, 0x1U, 0, 0, 0x400E82A0U
- #define **IOMUXC_GPIO_EMC_B1_19_TMR3_TIMER0** 0x400E805CU, 0x2U, 0x400E8654U, 0x0U, 0x400E82A0U
- #define **IOMUXC_GPIO_EMC_B1_19_GPIO_MUX1_IO19** 0x400E805CU, 0x5U, 0, 0, 0x400-E82A0U
- #define **IOMUXC_GPIO_EMC_B1_19_FLEXIO1_D19** 0x400E805CU, 0x8U, 0, 0, 0x400E82-A0U
- #define **IOMUXC_GPIO_EMC_B1_19_GPIO7_IO19** 0x400E805CU, 0xAU, 0, 0, 0x400E82-A0U
- #define **IOMUXC_GPIO_EMC_B1_20_SEMC_ADDR12** 0x400E8060U, 0x0U, 0, 0, 0x400-E82A4U
- #define **IOMUXC_GPIO_EMC_B1_20_FLEXPWM2_PWM3_B** 0x400E8060U, 0x1U, 0, 0, 0x400E82A4U
- #define **IOMUXC_GPIO_EMC_B1_20_TMR4_TIMER0** 0x400E8060U, 0x2U, 0x400E8660U, 0x0U, 0x400E82A4U
- #define **IOMUXC_GPIO_EMC_B1_20_GPIO_MUX1_IO20** 0x400E8060U, 0x5U, 0, 0, 0x400-E82A4U
- #define **IOMUXC_GPIO_EMC_B1_20_FLEXIO1_D20** 0x400E8060U, 0x8U, 0, 0, 0x400E82-A4U
- #define **IOMUXC_GPIO_EMC_B1_20_GPIO7_IO20** 0x400E8060U, 0xAU, 0, 0, 0x400E82A4-U
- #define **IOMUXC_GPIO_EMC_B1_21_GPIO7_IO21** 0x400E8064U, 0xAU, 0, 0, 0x400E82A8-U
- #define **IOMUXC_GPIO_EMC_B1_21_SEMC_BA0** 0x400E8064U, 0x0U, 0, 0, 0x400E82A8U
- #define **IOMUXC_GPIO_EMC_B1_21_FLEXPWM3_PWM3_A** 0x400E8064U, 0x1U, 0x400-E853CU, 0x0U, 0x400E82A8U
- #define **IOMUXC_GPIO_EMC_B1_21_GPIO_MUX1_IO21** 0x400E8064U, 0x5U, 0, 0, 0x400-E82A8U
- #define **IOMUXC_GPIO_EMC_B1_21_FLEXIO1_D21** 0x400E8064U, 0x8U, 0, 0, 0x400E82-A8U
- #define **IOMUXC_GPIO_EMC_B1_22_GPIO7_IO22** 0x400E8068U, 0xAU, 0, 0, 0x400E82A-CU
- #define **IOMUXC_GPIO_EMC_B1_22_SEMC_BA1** 0x400E8068U, 0x0U, 0, 0, 0x400E82ACU
- #define **IOMUXC_GPIO_EMC_B1_22_FLEXPWM3_PWM3_B** 0x400E8068U, 0x1U, 0x400-E854CU, 0x0U, 0x400E82ACU
- #define **IOMUXC_GPIO_EMC_B1_22_GPIO_MUX1_IO22** 0x400E8068U, 0x5U, 0, 0, 0x400-E82ACU
- #define **IOMUXC_GPIO_EMC_B1_22_FLEXIO1_D22** 0x400E8068U, 0x8U, 0, 0, 0x400E82-ACU
- #define **IOMUXC_GPIO_EMC_B1_23_SEMC_ADDR10** 0x400E806CU, 0x0U, 0, 0, 0x400-E82B0U
- #define **IOMUXC_GPIO_EMC_B1_23_FLEXPWM1_PWM0_A** 0x400E806CU, 0x1U, 0x400-E8500U, 0x0U, 0x400E82B0U

- #define **IOMUXC_GPIO_EMC_B1_23_GPIO_MUX1_IO23** 0x400E806CU, 0x5U, 0, 0, 0x400E82B0U
- #define **IOMUXC_GPIO_EMC_B1_23_FLEXIO1_D23** 0x400E806CU, 0x8U, 0, 0, 0x400E82B0U
- #define **IOMUXC_GPIO_EMC_B1_23_GPIO7_IO23** 0x400E806CU, 0xAU, 0, 0, 0x400E82B0U
- #define **IOMUXC_GPIO_EMC_B1_24_GPIO7_IO24** 0x400E8070U, 0xAU, 0, 0, 0x400E82B4U
- #define **IOMUXC_GPIO_EMC_B1_24_SEMC_CAS** 0x400E8070U, 0x0U, 0, 0, 0x400E82B4U
- #define **IOMUXC_GPIO_EMC_B1_24_FLEXPWM1_PWM0_B** 0x400E8070U, 0x1U, 0x400E850CU, 0x0U, 0x400E82B4U
- #define **IOMUXC_GPIO_EMC_B1_24_GPIO_MUX1_IO24** 0x400E8070U, 0x5U, 0, 0, 0x400E82B4U
- #define **IOMUXC_GPIO_EMC_B1_24_FLEXIO1_D24** 0x400E8070U, 0x8U, 0, 0, 0x400E82B4U
- #define **IOMUXC_GPIO_EMC_B1_25_GPIO7_IO25** 0x400E8074U, 0xAU, 0, 0, 0x400E82B8U
- #define **IOMUXC_GPIO_EMC_B1_25_SEMC_RAS** 0x400E8074U, 0x0U, 0, 0, 0x400E82B8U
- #define **IOMUXC_GPIO_EMC_B1_25_FLEXPWM1_PWM1_A** 0x400E8074U, 0x1U, 0x400E8504U, 0x0U, 0x400E82B8U
- #define **IOMUXC_GPIO_EMC_B1_25_GPIO_MUX1_IO25** 0x400E8074U, 0x5U, 0, 0, 0x400E82B8U
- #define **IOMUXC_GPIO_EMC_B1_25_FLEXIO1_D25** 0x400E8074U, 0x8U, 0, 0, 0x400E82B8U
- #define **IOMUXC_GPIO_EMC_B1_26_SEMC_CLK** 0x400E8078U, 0x0U, 0, 0, 0x400E82BCU
- #define **IOMUXC_GPIO_EMC_B1_26_FLEXPWM1_PWM1_B** 0x400E8078U, 0x1U, 0x400E8510U, 0x0U, 0x400E82BCU
- #define **IOMUXC_GPIO_EMC_B1_26_GPIO_MUX1_IO26** 0x400E8078U, 0x5U, 0, 0, 0x400E82BCU
- #define **IOMUXC_GPIO_EMC_B1_26_FLEXIO1_D26** 0x400E8078U, 0x8U, 0, 0, 0x400E82BCU
- #define **IOMUXC_GPIO_EMC_B1_26_GPIO7_IO26** 0x400E8078U, 0xAU, 0, 0, 0x400E82BCU
- #define **IOMUXC_GPIO_EMC_B1_27_GPIO7_IO27** 0x400E807CU, 0xAU, 0, 0, 0x400E82C0U
- #define **IOMUXC_GPIO_EMC_B1_27_SEMC_CKE** 0x400E807CU, 0x0U, 0, 0, 0x400E82C0U
- #define **IOMUXC_GPIO_EMC_B1_27_FLEXPWM1_PWM2_A** 0x400E807CU, 0x1U, 0x400E8508U, 0x0U, 0x400E82C0U
- #define **IOMUXC_GPIO_EMC_B1_27_GPIO_MUX1_IO27** 0x400E807CU, 0x5U, 0, 0, 0x400E82C0U
- #define **IOMUXC_GPIO_EMC_B1_27_FLEXIO1_D27** 0x400E807CU, 0x8U, 0, 0, 0x400E82C0U
- #define **IOMUXC_GPIO_EMC_B1_28_GPIO7_IO28** 0x400E8080U, 0xAU, 0, 0, 0x400E82C4U
- #define **IOMUXC_GPIO_EMC_B1_28_SEMC_WE** 0x400E8080U, 0x0U, 0, 0, 0x400E82C4U
- #define **IOMUXC_GPIO_EMC_B1_28_FLEXPWM1_PWM2_B** 0x400E8080U, 0x1U, 0x400E8514U, 0x0U, 0x400E82C4U
- #define **IOMUXC_GPIO_EMC_B1_28_GPIO_MUX1_IO28** 0x400E8080U, 0x5U, 0, 0, 0x400E82C4U

- E82C4U
- #define **IOMUXC_GPIO_EMC_B1_28_FLEXIO1_D28** 0x400E8080U, 0x8U, 0, 0, 0x400E82C4U
- #define **IOMUXC_GPIO_EMC_B1_29_SEMC_CS0** 0x400E8084U, 0x0U, 0, 0, 0x400E82C8U
- #define **IOMUXC_GPIO_EMC_B1_29_FLEXPWM3_PWM0_A** 0x400E8084U, 0x1U, 0x400E8530U, 0x0U, 0x400E82C8U
- #define **IOMUXC_GPIO_EMC_B1_29_GPIO_MUX1_IO29** 0x400E8084U, 0x5U, 0, 0, 0x400E82C8U
- #define **IOMUXC_GPIO_EMC_B1_29_FLEXIO1_D29** 0x400E8084U, 0x8U, 0, 0, 0x400E82C8U
- #define **IOMUXC_GPIO_EMC_B1_29_GPIO7_IO29** 0x400E8084U, 0xAU, 0, 0, 0x400E82C8U
- #define **IOMUXC_GPIO_EMC_B1_30_SEMC_DATA08** 0x400E8088U, 0x0U, 0, 0, 0x400E82CCU
- #define **IOMUXC_GPIO_EMC_B1_30_FLEXPWM3_PWM0_B** 0x400E8088U, 0x1U, 0x400E8540U, 0x0U, 0x400E82CCU
- #define **IOMUXC_GPIO_EMC_B1_30_GPIO_MUX1_IO30** 0x400E8088U, 0x5U, 0, 0, 0x400E82CCU
- #define **IOMUXC_GPIO_EMC_B1_30_FLEXIO1_D30** 0x400E8088U, 0x8U, 0, 0, 0x400E82CCU
- #define **IOMUXC_GPIO_EMC_B1_30_GPIO7_IO30** 0x400E8088U, 0xAU, 0, 0, 0x400E82CCU
- #define **IOMUXC_GPIO_EMC_B1_31_GPIO7_IO31** 0x400E808CU, 0xAU, 0, 0, 0x400E82D0U
- #define **IOMUXC_GPIO_EMC_B1_31_SEMC_DATA09** 0x400E808CU, 0x0U, 0, 0, 0x400E82D0U
- #define **IOMUXC_GPIO_EMC_B1_31_FLEXPWM3_PWM1_A** 0x400E808CU, 0x1U, 0x400E8534U, 0x0U, 0x400E82D0U
- #define **IOMUXC_GPIO_EMC_B1_31_GPIO_MUX1_IO31** 0x400E808CU, 0x5U, 0, 0, 0x400E82D0U
- #define **IOMUXC_GPIO_EMC_B1_31_FLEXIO1_D31** 0x400E808CU, 0x8U, 0, 0, 0x400E82D0U
- #define **IOMUXC_GPIO_EMC_B1_32_GPIO8_IO00** 0x400E8090U, 0xAU, 0, 0, 0x400E82D4U
- #define **IOMUXC_GPIO_EMC_B1_32_SEMC_DATA10** 0x400E8090U, 0x0U, 0, 0, 0x400E82D4U
- #define **IOMUXC_GPIO_EMC_B1_32_FLEXPWM3_PWM1_B** 0x400E8090U, 0x1U, 0x400E8544U, 0x0U, 0x400E82D4U
- #define **IOMUXC_GPIO_EMC_B1_32_GPIO_MUX2_IO00** 0x400E8090U, 0x5U, 0, 0, 0x400E82D4U
- #define **IOMUXC_GPIO_EMC_B1_33_SEMC_DATA11** 0x400E8094U, 0x0U, 0, 0, 0x400E82D8U
- #define **IOMUXC_GPIO_EMC_B1_33_FLEXPWM3_PWM2_A** 0x400E8094U, 0x1U, 0x400E8538U, 0x0U, 0x400E82D8U
- #define **IOMUXC_GPIO_EMC_B1_33_GPIO_MUX2_IO01** 0x400E8094U, 0x5U, 0, 0, 0x400E82D8U
- #define **IOMUXC_GPIO_EMC_B1_33_GPIO8_IO01** 0x400E8094U, 0xAU, 0, 0, 0x400E82D8U
- #define **IOMUXC_GPIO_EMC_B1_34_GPIO8_IO02** 0x400E8098U, 0xAU, 0, 0, 0x400E82DCU

- #define **IOMUXC_GPIO_EMC_B1_34_SEMC_DATA12** 0x400E8098U, 0x0U, 0, 0, 0x400E82-DCU
- #define **IOMUXC_GPIO_EMC_B1_34_FLEXPWM3_PWM2_B** 0x400E8098U, 0x1U, 0x400-E8548U, 0x0U, 0x400E82DCU
- #define **IOMUXC_GPIO_EMC_B1_34_GPIO_MUX2_IO02** 0x400E8098U, 0x5U, 0, 0, 0x400-E82DCU
- #define **IOMUXC_GPIO_EMC_B1_35_GPIO8_IO03** 0x400E809CU, 0xAU, 0, 0, 0x400E82-E0U
- #define **IOMUXC_GPIO_EMC_B1_35_SEMC_DATA13** 0x400E809CU, 0x0U, 0, 0, 0x400-E82E0U
- #define **IOMUXC_GPIO_EMC_B1_35_XBAR1_INOUT09** 0x400E809CU, 0x1U, 0, 0, 0x400-E82E0U
- #define **IOMUXC_GPIO_EMC_B1_35_GPIO_MUX2_IO03** 0x400E809CU, 0x5U, 0, 0, 0x400-E82E0U
- #define **IOMUXC_GPIO_EMC_B1_36_SEMC_DATA14** 0x400E80A0U, 0x0U, 0, 0, 0x400-E82E4U
- #define **IOMUXC_GPIO_EMC_B1_36_XBAR1_INOUT10** 0x400E80A0U, 0x1U, 0, 0, 0x400-E82E4U
- #define **IOMUXC_GPIO_EMC_B1_36_GPIO_MUX2_IO04** 0x400E80A0U, 0x5U, 0, 0, 0x400-E82E4U
- #define **IOMUXC_GPIO_EMC_B1_36_GPIO8_IO04** 0x400E80A0U, 0xAU, 0, 0, 0x400E82-E4U
- #define **IOMUXC_GPIO_EMC_B1_37_GPIO8_IO05** 0x400E80A4U, 0xAU, 0, 0, 0x400E82-E8U
- #define **IOMUXC_GPIO_EMC_B1_37_SEMC_DATA15** 0x400E80A4U, 0x0U, 0, 0, 0x400-E82E8U
- #define **IOMUXC_GPIO_EMC_B1_37_XBAR1_INOUT11** 0x400E80A4U, 0x1U, 0, 0, 0x400-E82E8U
- #define **IOMUXC_GPIO_EMC_B1_37_GPIO_MUX2_IO05** 0x400E80A4U, 0x5U, 0, 0, 0x400-E82E8U
- #define **IOMUXC_GPIO_EMC_B1_38_GPIO8_IO06** 0x400E80A8U, 0xAU, 0, 0, 0x400E82E-CU
- #define **IOMUXC_GPIO_EMC_B1_38_SEMC_DM01** 0x400E80A8U, 0x0U, 0, 0, 0x400E82E-CU
- #define **IOMUXC_GPIO_EMC_B1_38_FLEXPWM1_PWM3_A** 0x400E80A8U, 0x1U, 0, 0, 0x400E82ECU
- #define **IOMUXC_GPIO_EMC_B1_38_TMR1_TIMER1** 0x400E80A8U, 0x2U, 0x400E8640U, 0x0U, 0x400E82ECU
- #define **IOMUXC_GPIO_EMC_B1_38_GPIO_MUX2_IO06** 0x400E80A8U, 0x5U, 0, 0, 0x400-E82ECU
- #define **IOMUXC_GPIO_EMC_B1_39_SEMC_DQS** 0x400E80ACU, 0x0U, 0, 0, 0x400E82F0-U
- #define **IOMUXC_GPIO_EMC_B1_39_FLEXPWM1_PWM3_B** 0x400E80ACU, 0x1U, 0, 0, 0x400E82F0U
- #define **IOMUXC_GPIO_EMC_B1_39_TMR2_TIMER1** 0x400E80ACU, 0x2U, 0x400E864C-U, 0x0U, 0x400E82F0U
- #define **IOMUXC_GPIO_EMC_B1_39_GPIO_MUX2_IO07** 0x400E80ACU, 0x5U, 0, 0, 0x400E82F0U
- #define **IOMUXC_GPIO_EMC_B1_39_GPIO8_IO07** 0x400E80ACU, 0xAU, 0, 0, 0x400E82-

```

F0U
• #define IOMUXC_GPIO_EMC_B1_40_SEMC_RDY 0x400E80B0U, 0x0U, 0, 0, 0x400E82F4U
• #define IOMUXC_GPIO_EMC_B1_40_XBAR1_INOUT12 0x400E80B0U, 0x1U, 0, 0, 0x400-
E82F4U
• #define IOMUXC_GPIO_EMC_B1_40_MQS_RIGHT 0x400E80B0U, 0x2U, 0, 0, 0x400E82-
F4U
• #define IOMUXC_GPIO_EMC_B1_40_LPUART6_TXD 0x400E80B0U, 0x3U, 0, 0, 0x400-
E82F4U
• #define IOMUXC_GPIO_EMC_B1_40_GPIO_MUX2_IO08 0x400E80B0U, 0x5U, 0, 0, 0x400-
E82F4U
• #define IOMUXC_GPIO_EMC_B1_40_ENET_1G_MDC 0x400E80B0U, 0x7U, 0, 0, 0x400-
E82F4U
• #define IOMUXC_GPIO_EMC_B1_40_CCM_CLKO1 0x400E80B0U, 0x9U, 0, 0, 0x400E82-
F4U
• #define IOMUXC_GPIO_EMC_B1_40_GPIO8_IO08 0x400E80B0U, 0xAU, 0, 0, 0x400E82F4-
U
• #define IOMUXC_GPIO_EMC_B1_41_GPIO8_IO09 0x400E80B4U, 0xAU, 0, 0, 0x400E82F8-
U
• #define IOMUXC_GPIO_EMC_B1_41_SEMC_CSX00 0x400E80B4U, 0x0U, 0, 0, 0x400E82-
F8U
• #define IOMUXC_GPIO_EMC_B1_41_XBAR1_INOUT13 0x400E80B4U, 0x1U, 0, 0, 0x400-
E82F8U
• #define IOMUXC_GPIO_EMC_B1_41_MQS_LEFT 0x400E80B4U, 0x2U, 0, 0, 0x400E82F8U
• #define IOMUXC_GPIO_EMC_B1_41_LPUART6_RXD 0x400E80B4U, 0x3U, 0, 0, 0x400-
E82F8U
• #define IOMUXC_GPIO_EMC_B1_41_FLEXSPI2_B_DATA07 0x400E80B4U, 0x4U, 0, 0,
0x400E82F8U
• #define IOMUXC_GPIO_EMC_B1_41_GPIO_MUX2_IO09 0x400E80B4U, 0x5U, 0, 0, 0x400-
E82F8U
• #define IOMUXC_GPIO_EMC_B1_41_ENET_1G_MDIO 0x400E80B4U, 0x7U, 0x400E84C8-
U, 0x0U, 0x400E82F8U
• #define IOMUXC_GPIO_EMC_B1_41_CCM_CLKO2 0x400E80B4U, 0x9U, 0, 0, 0x400E82-
F8U
• #define IOMUXC_GPIO_EMC_B2_00_SEMC_DATA16 0x400E80B8U, 0x0U, 0, 0, 0x400-
E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_CCM_ENET_REF_CLK_25M 0x400E80B8U, 0x1U,
0, 0, 0x400E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_TMR3_TIMER1 0x400E80B8U, 0x2U, 0x400E8658U,
0x0U, 0x400E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_LPUART6_CTS_B 0x400E80B8U, 0x3U, 0, 0, 0x400-
E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_FLEXSPI2_B_DATA06 0x400E80B8U, 0x4U, 0, 0,
0x400E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_GPIO_MUX2_IO10 0x400E80B8U, 0x5U, 0, 0, 0x400-
E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_XBAR1_INOUT20 0x400E80B8U, 0x6U, 0x400E86-
D8U, 0x0U, 0x400E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_ENET_QOS_1588_EVENT1_OUT 0x400E80B8U,
0x7U, 0, 0, 0x400E82FCU
• #define IOMUXC_GPIO_EMC_B2_00_LPSPI1_SCK 0x400E80B8U, 0x8U, 0x400E85D0U,

```

- 0x0U, 0x400E82FCU
- #define **IOMUXC_GPIO_EMC_B2_00_LPI2C2_SCL** 0x400E80B8U, 0x9U, 0x400E85B4U, 0x0U, 0x400E82FCU
- #define **IOMUXC_GPIO_EMC_B2_00_GPIO8_IO10** 0x400E80B8U, 0xAU, 0, 0, 0x400E82FCU
- #define **IOMUXC_GPIO_EMC_B2_00_FLEXPWM3_PWM0_A** 0x400E80B8U, 0xBU, 0x400E8530U, 0x1U, 0x400E82FCU
- #define **IOMUXC_GPIO_EMC_B2_01_SEMC_DATA17** 0x400E80BCU, 0x0U, 0, 0, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_USDHC2_CD_B** 0x400E80BCU, 0x1U, 0x400E86D0U, 0x0U, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_TMR4_TIMER1** 0x400E80BCU, 0x2U, 0x400E8664U, 0x0U, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_LPUART6_RTS_B** 0x400E80BCU, 0x3U, 0, 0, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_FLEXSPI2_B_DATA05** 0x400E80BCU, 0x4U, 0, 0, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_GPIO_MUX2_IO11** 0x400E80BCU, 0x5U, 0, 0, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_XBAR1_INOUT21** 0x400E80BCU, 0x6U, 0x400E86DCU, 0x0U, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_ENET_QOS_1588_EVENT1_IN** 0x400E80BCU, 0x7U, 0, 0, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_LPSPI1_PCS0** 0x400E80BCU, 0x8U, 0x400E85CCU, 0x0U, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_LPI2C2_SDA** 0x400E80BCU, 0x9U, 0x400E85B8U, 0x0U, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_GPIO8_IO11** 0x400E80BCU, 0xAU, 0, 0, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_01_FLEXPWM3_PWM0_B** 0x400E80BCU, 0xBU, 0x400E8540U, 0x1U, 0x400E8300U
- #define **IOMUXC_GPIO_EMC_B2_02_SEMC_DATA18** 0x400E80C0U, 0x0U, 0, 0, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_USDHC2_WP** 0x400E80C0U, 0x1U, 0x400E86D4U, 0x0U, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_VIDEO_MUX_CSI_DATA23** 0x400E80C0U, 0x3U, 0, 0, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_FLEXSPI2_B_DATA04** 0x400E80C0U, 0x4U, 0, 0, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_GPIO_MUX2_IO12** 0x400E80C0U, 0x5U, 0, 0, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_XBAR1_INOUT22** 0x400E80C0U, 0x6U, 0x400E86E0U, 0x0U, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_ENET_QOS_1588_EVENT1_AUX_IN** 0x400E80C0U, 0x7U, 0, 0, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_LPSPI1_SOUT** 0x400E80C0U, 0x8U, 0x400E85D8U, 0x0U, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_GPIO8_IO12** 0x400E80C0U, 0xAU, 0, 0, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_02_FLEXPWM3_PWM1_A** 0x400E80C0U, 0xBU,

- 0x400E8534U, 0x1U, 0x400E8304U
- #define **IOMUXC_GPIO_EMC_B2_03_SEMC_DATA19** 0x400E80C4U, 0x0U, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_USDHC2_VSELECT** 0x400E80C4U, 0x1U, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_VIDEO_MUX_CSI_DATA22** 0x400E80C4U, 0x3U, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_FLEXSPI2_B_DATA03** 0x400E80C4U, 0x4U, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_GPIO_MUX2_IO13** 0x400E80C4U, 0x5U, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_XBAR1_INOUT23** 0x400E80C4U, 0x6U, 0x400E86E4U, 0x0U, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_ENET_1G_TX_DATA03** 0x400E80C4U, 0x7U, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_LPSPI1_SIN** 0x400E80C4U, 0x8U, 0x400E85D4U, 0x0U, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_GPIO8_IO13** 0x400E80C4U, 0xAU, 0, 0, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_03_FLEXPWM3_PWM1_B** 0x400E80C4U, 0xBU, 0x400E8544U, 0x1U, 0x400E8308U
- #define **IOMUXC_GPIO_EMC_B2_04_SEMC_DATA20** 0x400E80C8U, 0x0U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_USDHC2_RESET_B** 0x400E80C8U, 0x1U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_SAI2_MCLK** 0x400E80C8U, 0x2U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_VIDEO_MUX_CSI_DATA21** 0x400E80C8U, 0x3U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_FLEXSPI2_B_DATA02** 0x400E80C8U, 0x4U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_GPIO_MUX2_IO14** 0x400E80C8U, 0x5U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_XBAR1_INOUT24** 0x400E80C8U, 0x6U, 0x400E86E8U, 0x0U, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_ENET_1G_TX_DATA02** 0x400E80C8U, 0x7U, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_LPSPI3_SCK** 0x400E80C8U, 0x8U, 0x400E8600U, 0x0U, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_GPIO8_IO14** 0x400E80C8U, 0xAU, 0, 0, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_04_FLEXPWM3_PWM2_A** 0x400E80C8U, 0xBU, 0x400E8538U, 0x1U, 0x400E830CU
- #define **IOMUXC_GPIO_EMC_B2_05_SEMC_DATA21** 0x400E80CCU, 0x0U, 0, 0, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_GPT3_CLK** 0x400E80CCU, 0x1U, 0x400E8598U, 0x0U, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_SAI2_RX_SYNC** 0x400E80CCU, 0x2U, 0, 0, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_VIDEO_MUX_CSI_DATA20** 0x400E80CCU, 0x3U,

- 0, 0, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_FLEXSPI2_B_DATA01** 0x400E80CCU, 0x4U, 0, 0, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_GPIO_MUX2_IO15** 0x400E80CCU, 0x5U, 0, 0, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_XBAR1_INOUT25** 0x400E80CCU, 0x6U, 0x400E86E-CU, 0x0U, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_ENET_1G_RX_CLK** 0x400E80CCU, 0x7U, 0x400-E84CCU, 0x0U, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_LPSPI3_PCS0** 0x400E80CCU, 0x8U, 0x400E85F0U, 0x0U, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_PIT1_TRIGGER0** 0x400E80CCU, 0x9U, 0, 0, 0x400-E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_GPIO8_IO15** 0x400E80CCU, 0xAU, 0, 0, 0x400-E8310U
- #define **IOMUXC_GPIO_EMC_B2_05_FLEXPWM3_PWM2_B** 0x400E80CCU, 0xBU, 0x400E8548U, 0x1U, 0x400E8310U
- #define **IOMUXC_GPIO_EMC_B2_06_SEMC_DATA22** 0x400E80D0U, 0x0U, 0, 0, 0x400-E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_GPT3_CAPTURE1** 0x400E80D0U, 0x1U, 0x400-E8590U, 0x0U, 0x400E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_GPIO8_IO16** 0x400E80D0U, 0xAU, 0, 0, 0x400E8314-U
- #define **IOMUXC_GPIO_EMC_B2_06_SAI2_RX_BCLK** 0x400E80D0U, 0x2U, 0, 0, 0x400-E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_FLEXPWM3_PWM3_A** 0x400E80D0U, 0xBU, 0x400E853CU, 0x1U, 0x400E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_VIDEO_MUX_CSI_DATA19** 0x400E80D0U, 0x3U, 0, 0, 0x400E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_FLEXSPI2_B_DATA00** 0x400E80D0U, 0x4U, 0, 0, 0x400E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_GPIO_MUX2_IO16** 0x400E80D0U, 0x5U, 0, 0, 0x400-E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_XBAR1_INOUT26** 0x400E80D0U, 0x6U, 0x400E86-F0U, 0x0U, 0x400E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_ENET_1G_TX_ER** 0x400E80D0U, 0x7U, 0, 0, 0x400-E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_LPSPI3_SOUT** 0x400E80D0U, 0x8U, 0x400E8608U, 0x0U, 0x400E8314U
- #define **IOMUXC_GPIO_EMC_B2_06_PIT1_TRIGGER1** 0x400E80D0U, 0x9U, 0, 0, 0x400-E8314U
- #define **IOMUXC_GPIO_EMC_B2_07_SEMC_DATA23** 0x400E80D4U, 0x0U, 0, 0, 0x400-E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_GPT3_CAPTURE2** 0x400E80D4U, 0x1U, 0x400-E8594U, 0x0U, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_SAI2_RX_DATA** 0x400E80D4U, 0x2U, 0, 0, 0x400-E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_VIDEO_MUX_CSI_DATA18** 0x400E80D4U, 0x3U, 0, 0, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_FLEXSPI2_B_DQS** 0x400E80D4U, 0x4U, 0, 0, 0x400-

- E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_GPIO_MUX2_IO17** 0x400E80D4U, 0x5U, 0, 0, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_XBAR1_INOUT27** 0x400E80D4U, 0x6U, 0x400E86F4U, 0x0U, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_ENET_1G_RX_DATA03** 0x400E80D4U, 0x7U, 0x400E84DCU, 0x0U, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_LPSPI3_SIN** 0x400E80D4U, 0x8U, 0x400E8604U, 0x0U, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_PIT1_TRIGGER2** 0x400E80D4U, 0x9U, 0, 0, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_GPIO8_IO17** 0x400E80D4U, 0xAU, 0, 0, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_07_FLEXPWM3_PWM3_B** 0x400E80D4U, 0xBU, 0x400E854CU, 0x1U, 0x400E8318U
- #define **IOMUXC_GPIO_EMC_B2_08_SEMC_DM02** 0x400E80D8U, 0x0U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_GPT3_COMPARE1** 0x400E80D8U, 0x1U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_SAI2_TX_DATA** 0x400E80D8U, 0x2U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_VIDEO_MUX_CSI_DATA17** 0x400E80D8U, 0x3U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_FLEXSPI2_B_SS0_B** 0x400E80D8U, 0x4U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_GPIO_MUX2_IO18** 0x400E80D8U, 0x5U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_XBAR1_INOUT28** 0x400E80D8U, 0x6U, 0x400E86F8U, 0x0U, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_ENET_1G_RX_DATA02** 0x400E80D8U, 0x7U, 0x400E84D8U, 0x0U, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_LPSPI3_PCS1** 0x400E80D8U, 0x8U, 0x400E85F4U, 0x0U, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_PIT1_TRIGGER3** 0x400E80D8U, 0x9U, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_08_GPIO8_IO18** 0x400E80D8U, 0xAU, 0, 0, 0x400E831CU
- #define **IOMUXC_GPIO_EMC_B2_09_GPIO8_IO19** 0x400E80DCU, 0xAU, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_SEMC_DATA24** 0x400E80DCU, 0x0U, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_GPT3_COMPARE2** 0x400E80DCU, 0x1U, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_SAI2_TX_BCLK** 0x400E80DCU, 0x2U, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_VIDEO_MUX_CSI_DATA16** 0x400E80DCU, 0x3U, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_FLEXSPI2_B_SCLK** 0x400E80DCU, 0x4U, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_GPIO_MUX2_IO19** 0x400E80DCU, 0x5U, 0, 0,

- 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_XBAR1_INOUT29** 0x400E80DCU, 0x6U, 0x400E86FCU, 0x0U, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_ENET_1G_CRS** 0x400E80DCU, 0x7U, 0, 0, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_LPSPI3_PCS2** 0x400E80DCU, 0x8U, 0x400E85F8U, 0x0U, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_09_TMR1_TIMER0** 0x400E80DCU, 0x9U, 0x400E863CU, 0x1U, 0x400E8320U
- #define **IOMUXC_GPIO_EMC_B2_10_GPIO8_IO20** 0x400E80E0U, 0xAU, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_SEMC_DATA25** 0x400E80E0U, 0x0U, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_GPT3_COMPARE3** 0x400E80E0U, 0x1U, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_SAI2_TX_SYNC** 0x400E80E0U, 0x2U, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_VIDEO_MUX_CSI_FIELD** 0x400E80E0U, 0x3U, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_FLEXSPI2_A_SCLK** 0x400E80E0U, 0x4U, 0x400E858CU, 0x0U, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_GPIO_MUX2_IO20** 0x400E80E0U, 0x5U, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_XBAR1_INOUT30** 0x400E80E0U, 0x6U, 0x400E8700U, 0x0U, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_ENET_1G_COL** 0x400E80E0U, 0x7U, 0, 0, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_LPSPI3_PCS3** 0x400E80E0U, 0x8U, 0x400E85FCU, 0x0U, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_10_TMR1_TIMER1** 0x400E80E0U, 0x9U, 0x400E8640U, 0x1U, 0x400E8324U
- #define **IOMUXC_GPIO_EMC_B2_11_SEMC_DATA26** 0x400E80E4U, 0x0U, 0, 0, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_SPDIF_IN** 0x400E80E4U, 0x1U, 0x400E86B4U, 0x0U, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_ENET_1G_TX_DATA00** 0x400E80E4U, 0x2U, 0, 0, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_SAI3_RX_SYNC** 0x400E80E4U, 0x3U, 0, 0, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_FLEXSPI2_A_SS0_B** 0x400E80E4U, 0x4U, 0, 0, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_GPIO_MUX2_IO21** 0x400E80E4U, 0x5U, 0, 0, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_XBAR1_INOUT31** 0x400E80E4U, 0x6U, 0x400E8704U, 0x0U, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_EMVSIM1_IO** 0x400E80E4U, 0x8U, 0x400E869CU, 0x0U, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_TMR1_TIMER2** 0x400E80E4U, 0x9U, 0x400E8644U, 0x0U, 0x400E8328U
- #define **IOMUXC_GPIO_EMC_B2_11_GPIO8_IO21** 0x400E80E4U, 0xAU, 0, 0, 0x400E8328U

- U
- #define **IOMUXC_GPIO_EMC_B2_12_SEMC_DATA27** 0x400E80E8U, 0x0U, 0, 0, 0x400-E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_SPDIF_OUT** 0x400E80E8U, 0x1U, 0, 0, 0x400E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_ENET_1G_TX_DATA01** 0x400E80E8U, 0x2U, 0, 0, 0x400E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_SAI3_RX_BCLK** 0x400E80E8U, 0x3U, 0, 0, 0x400-E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_FLEXSPI2_A_DQS** 0x400E80E8U, 0x4U, 0, 0, 0x400-E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_GPIO_MUX2_IO22** 0x400E80E8U, 0x5U, 0, 0, 0x400-E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_XBAR1_INOUT32** 0x400E80E8U, 0x6U, 0x400-E8708U, 0x0U, 0x400E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_EMVSIM1_CLK** 0x400E80E8U, 0x8U, 0, 0, 0x400-E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_TMR1_TIMER3** 0x400E80E8U, 0x9U, 0, 0, 0x400-E832CU
 - #define **IOMUXC_GPIO_EMC_B2_12_GPIO8_IO22** 0x400E80E8U, 0xAU, 0, 0, 0x400E832-CU
 - #define **IOMUXC_GPIO_EMC_B2_13_GPIO8_IO23** 0x400E80ECU, 0xAU, 0, 0, 0x400E8330-U
 - #define **IOMUXC_GPIO_EMC_B2_13_SEMC_DATA28** 0x400E80ECU, 0x0U, 0, 0, 0x400-E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_ENET_1G_TX_EN** 0x400E80ECU, 0x2U, 0, 0, 0x400-E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_SAI3_RX_DATA** 0x400E80ECU, 0x3U, 0, 0, 0x400-E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_FLEXSPI2_A_DATA00** 0x400E80ECU, 0x4U, 0x400-E857CU, 0x0U, 0x400E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_GPIO_MUX2_IO23** 0x400E80ECU, 0x5U, 0, 0, 0x400E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_XBAR1_INOUT33** 0x400E80ECU, 0x6U, 0x400E870-CU, 0x0U, 0x400E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_EMVSIM1_RST** 0x400E80ECU, 0x8U, 0, 0, 0x400-E8330U
 - #define **IOMUXC_GPIO_EMC_B2_13_TMR2_TIMER0** 0x400E80ECU, 0x9U, 0x400E8648U, 0x1U, 0x400E8330U
 - #define **IOMUXC_GPIO_EMC_B2_14_SEMC_DATA29** 0x400E80F0U, 0x0U, 0, 0, 0x400-E8334U
 - #define **IOMUXC_GPIO_EMC_B2_14_ENET_1G_TX_CLK_IO** 0x400E80F0U, 0x2U, 0x400-E84E8U, 0x0U, 0x400E8334U
 - #define **IOMUXC_GPIO_EMC_B2_14_SAI3_TX_DATA** 0x400E80F0U, 0x3U, 0, 0, 0x400-E8334U
 - #define **IOMUXC_GPIO_EMC_B2_14_FLEXSPI2_A_DATA01** 0x400E80F0U, 0x4U, 0x400-E8580U, 0x0U, 0x400E8334U
 - #define **IOMUXC_GPIO_EMC_B2_14_GPIO_MUX2_IO24** 0x400E80F0U, 0x5U, 0, 0, 0x400-E8334U
 - #define **IOMUXC_GPIO_EMC_B2_14_XBAR1_INOUT34** 0x400E80F0U, 0x6U, 0x400E8710-U, 0x0U, 0x400E8334U

- #define **IOMUXC_GPIO_EMC_B2_14_SFA_ipp_do_atx_clk_under_test** 0x400E80F0U, 0x7U, 0, 0, 0x400E8334U
- #define **IOMUXC_GPIO_EMC_B2_14_EMVSIM1_SVEN** 0x400E80F0U, 0x8U, 0, 0, 0x400E8334U
- #define **IOMUXC_GPIO_EMC_B2_14_TMR2_TIMER1** 0x400E80F0U, 0x9U, 0x400E864CU, 0x1U, 0x400E8334U
- #define **IOMUXC_GPIO_EMC_B2_14_GPIO8_IO24** 0x400E80F0U, 0xAU, 0, 0, 0x400E8334U
- #define **IOMUXC_GPIO_EMC_B2_15_SEMC_DATA30** 0x400E80F4U, 0x0U, 0, 0, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_ENET_1G_RX_DATA00** 0x400E80F4U, 0x2U, 0x400E84D0U, 0x0U, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_SAI3_TX_BCLK** 0x400E80F4U, 0x3U, 0, 0, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_FLEXSPI2_A_DATA02** 0x400E80F4U, 0x4U, 0x400E8584U, 0x0U, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_GPIO_MUX2_IO25** 0x400E80F4U, 0x5U, 0, 0, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_XBAR1_INOUT35** 0x400E80F4U, 0x6U, 0x400E8714U, 0x0U, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_EMVSIM1_PD** 0x400E80F4U, 0x8U, 0x400E86A0U, 0x0U, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_TMR2_TIMER2** 0x400E80F4U, 0x9U, 0x400E8650U, 0x0U, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_15_GPIO8_IO25** 0x400E80F4U, 0xAU, 0, 0, 0x400E8338U
- #define **IOMUXC_GPIO_EMC_B2_16_GPIO8_IO26** 0x400E80F8U, 0xAU, 0, 0, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_SEMC_DATA31** 0x400E80F8U, 0x0U, 0, 0, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_XBAR1_INOUT14** 0x400E80F8U, 0x1U, 0, 0, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_ENET_1G_RX_DATA01** 0x400E80F8U, 0x2U, 0x400E84D4U, 0x0U, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_SAI3_TX_SYNC** 0x400E80F8U, 0x3U, 0, 0, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_FLEXSPI2_A_DATA03** 0x400E80F8U, 0x4U, 0x400E8588U, 0x0U, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_GPIO_MUX2_IO26** 0x400E80F8U, 0x5U, 0, 0, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_EMVSIM1_POWER_FAIL** 0x400E80F8U, 0x8U, 0x400E86A4U, 0x0U, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_16_TMR2_TIMER3** 0x400E80F8U, 0x9U, 0, 0, 0x400E833CU
- #define **IOMUXC_GPIO_EMC_B2_17_SEMC_DM03** 0x400E80FCU, 0x0U, 0, 0, 0x400E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_XBAR1_INOUT15** 0x400E80FCU, 0x1U, 0, 0, 0x400E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_ENET_1G_RX_EN** 0x400E80FCU, 0x2U, 0x400E84-

- E0U, 0x0U, 0x400E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_SAI3_MCLK** 0x400E80FCU, 0x3U, 0, 0, 0x400E8340-U
- #define **IOMUXC_GPIO_EMC_B2_17_FLEXSPI2_A_DATA04** 0x400E80FCU, 0x4U, 0, 0, 0x400E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_GPIO_MUX2_IO27** 0x400E80FCU, 0x5U, 0, 0, 0x400-E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_WDOG1_ANY** 0x400E80FCU, 0x8U, 0, 0, 0x400-E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_TMR3_TIMER0** 0x400E80FCU, 0x9U, 0x400E8654U, 0x1U, 0x400E8340U
- #define **IOMUXC_GPIO_EMC_B2_17_GPIO8_IO27** 0x400E80FCU, 0xAU, 0, 0, 0x400E8340-U
- #define **IOMUXC_GPIO_EMC_B2_18_SEMC_DQS4** 0x400E8100U, 0x0U, 0, 0, 0x400E8344-U
- #define **IOMUXC_GPIO_EMC_B2_18_XBAR1_INOUT16** 0x400E8100U, 0x1U, 0, 0, 0x400-E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_ENET_1G_RX_ER** 0x400E8100U, 0x2U, 0x400E84-E4U, 0x0U, 0x400E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_EWM_OUT_B** 0x400E8100U, 0x3U, 0, 0, 0x400-E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_FLEXSPI2_A_DATA05** 0x400E8100U, 0x4U, 0, 0, 0x400E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_GPIO_MUX2_IO28** 0x400E8100U, 0x5U, 0, 0, 0x400-E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_FLEXSPI1_A_DQS** 0x400E8100U, 0x6U, 0x400-E8550U, 0x0U, 0x400E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_WDOG1_B** 0x400E8100U, 0x8U, 0, 0, 0x400E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_TMR3_TIMER1** 0x400E8100U, 0x9U, 0x400E8658U, 0x1U, 0x400E8344U
- #define **IOMUXC_GPIO_EMC_B2_18_GPIO8_IO28** 0x400E8100U, 0xAU, 0, 0, 0x400E8344-U
- #define **IOMUXC_GPIO_EMC_B2_19_GPIO8_IO29** 0x400E8104U, 0xAU, 0, 0, 0x400E8348-U
- #define **IOMUXC_GPIO_EMC_B2_19_SEMC_CLKX00** 0x400E8104U, 0x0U, 0, 0, 0x400-E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_ENET_MDC** 0x400E8104U, 0x1U, 0, 0, 0x400E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_ENET_1G_MDC** 0x400E8104U, 0x2U, 0, 0, 0x400-E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_ENET_1G_REF_CLK** 0x400E8104U, 0x3U, 0x400-E84C4U, 0x0U, 0x400E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_FLEXSPI2_A_DATA06** 0x400E8104U, 0x4U, 0, 0, 0x400E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_GPIO_MUX2_IO29** 0x400E8104U, 0x5U, 0, 0, 0x400-E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_ENET_QOS_MDC** 0x400E8104U, 0x8U, 0, 0, 0x400-E8348U
- #define **IOMUXC_GPIO_EMC_B2_19_TMR3_TIMER2** 0x400E8104U, 0x9U, 0x400E865CU, 0x0U, 0x400E8348U
- #define **IOMUXC_GPIO_EMC_B2_20_GPIO8_IO30** 0x400E8108U, 0xAU, 0, 0, 0x400E834-

- CU
- #define **IOMUXC_GPIO_EMC_B2_20_SEMC_CLKX01** 0x400E8108U, 0x0U, 0, 0, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_ENET_MDIO** 0x400E8108U, 0x1U, 0x400E84ACU, 0x0U, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_ENET_1G_MDIO** 0x400E8108U, 0x2U, 0x400E84C8U, 0x1U, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_ENET_QOS_REF_CLK** 0x400E8108U, 0x3U, 0x400E84A0U, 0x0U, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_FLEXSPI2_A_DATA07** 0x400E8108U, 0x4U, 0, 0, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_GPIO_MUX2_IO30** 0x400E8108U, 0x5U, 0, 0, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_ENET_QOS_MDIO** 0x400E8108U, 0x8U, 0x400E84ECU, 0x0U, 0x400E834CU
- #define **IOMUXC_GPIO_EMC_B2_20_TMR3_TIMER3** 0x400E8108U, 0x9U, 0, 0, 0x400E834CU
- #define **IOMUXC_GPIO_AD_00_GPIO8_IO31** 0x400E810CU, 0xAU, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_EMVSI1_IO** 0x400E810CU, 0x0U, 0x400E869CU, 0x1U, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_FLEXCAN2_TX** 0x400E810CU, 0x1U, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_ENET_1G_1588_EVENT1_IN** 0x400E810CU, 0x2U, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_GPT2_CAPTURE1** 0x400E810CU, 0x3U, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_FLEXPWM1_PWM0_A** 0x400E810CU, 0x4U, 0x400E8500U, 0x1U, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_GPIO_MUX2_IO31** 0x400E810CU, 0x5U, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_LPUART7_TXD** 0x400E810CU, 0x6U, 0x400E8630U, 0x0U, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_FLEXIO2_D00** 0x400E810CU, 0x8U, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_00_FLEXSPI2_B_SS1_B** 0x400E810CU, 0x9U, 0, 0, 0x400E8350U
- #define **IOMUXC_GPIO_AD_01_GPIO9_IO00** 0x400E8110U, 0xAU, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_EMVSI1_CLK** 0x400E8110U, 0x0U, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_FLEXCAN2_RX** 0x400E8110U, 0x1U, 0x400E849CU, 0x0U, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_ENET_1G_1588_EVENT1_OUT** 0x400E8110U, 0x2U, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_GPT2_CAPTURE2** 0x400E8110U, 0x3U, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_FLEXPWM1_PWM0_B** 0x400E8110U, 0x4U, 0x400E850CU, 0x1U, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_GPIO_MUX3_IO00** 0x400E8110U, 0x5U, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_LPUART7_RXD** 0x400E8110U, 0x6U, 0x400E862CU, 0x0U, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_FLEXIO2_D01** 0x400E8110U, 0x8U, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_01_FLEXSPI2_A_SS1_B** 0x400E8110U, 0x9U, 0, 0, 0x400E8354U
- #define **IOMUXC_GPIO_AD_02_GPIO9_IO01** 0x400E8114U, 0xAU, 0, 0, 0x400E8358U

- #define **IOMUXC_GPIO_AD_02_EMVSIM1_RST** 0x400E8114U, 0x0U, 0, 0, 0x400E8358U
- #define **IOMUXC_GPIO_AD_02_LPUART7_CTS_B** 0x400E8114U, 0x1U, 0, 0, 0x400E8358U
- #define **IOMUXC_GPIO_AD_02_ENET_1G_1588_EVENT2_IN** 0x400E8114U, 0x2U, 0, 0, 0x400E8358U
- #define **IOMUXC_GPIO_AD_02_GPT2_COMPARE1** 0x400E8114U, 0x3U, 0, 0, 0x400E8358-U
- #define **IOMUXC_GPIO_AD_02_FLEXPWM1_PWM1_A** 0x400E8114U, 0x4U, 0x400E8504-U, 0x1U, 0x400E8358U
- #define **IOMUXC_GPIO_AD_02_GPIO_MUX3_IO01** 0x400E8114U, 0x5U, 0, 0, 0x400E8358-U
- #define **IOMUXC_GPIO_AD_02_LPUART8_TXD** 0x400E8114U, 0x6U, 0x400E8638U, 0x0U, 0x400E8358U
- #define **IOMUXC_GPIO_AD_02_FLEXIO2_D02** 0x400E8114U, 0x8U, 0, 0, 0x400E8358U
- #define **IOMUXC_GPIO_AD_02_VIDEO_MUX_EXT_DCIC1** 0x400E8114U, 0x9U, 0, 0, 0x400E8358U
- #define **IOMUXC_GPIO_AD_03_GPIO9_IO02** 0x400E8118U, 0xAU, 0, 0, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_EMVSIM1_SVEN** 0x400E8118U, 0x0U, 0, 0, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_LPUART7_RTS_B** 0x400E8118U, 0x1U, 0, 0, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_ENET_1G_1588_EVENT2_OUT** 0x400E8118U, 0x2U, 0, 0, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_GPT2_COMPARE2** 0x400E8118U, 0x3U, 0, 0, 0x400E835-CU
- #define **IOMUXC_GPIO_AD_03_FLEXPWM1_PWM1_B** 0x400E8118U, 0x4U, 0x400E8510-U, 0x1U, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_GPIO_MUX3_IO02** 0x400E8118U, 0x5U, 0, 0, 0x400E835-CU
- #define **IOMUXC_GPIO_AD_03_LPUART8_RXD** 0x400E8118U, 0x6U, 0x400E8634U, 0x0U, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_FLEXIO2_D03** 0x400E8118U, 0x8U, 0, 0, 0x400E835CU
- #define **IOMUXC_GPIO_AD_03_VIDEO_MUX_EXT_DCIC2** 0x400E8118U, 0x9U, 0, 0, 0x400E835CU
- #define **IOMUXC_GPIO_AD_04_EMVSIM1_PD** 0x400E811CU, 0x0U, 0x400E86A0U, 0x1U, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_LPUART8_CTS_B** 0x400E811CU, 0x1U, 0, 0, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_ENET_1G_1588_EVENT3_IN** 0x400E811CU, 0x2U, 0, 0, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_GPT2_COMPARE3** 0x400E811CU, 0x3U, 0, 0, 0x400-E8360U
- #define **IOMUXC_GPIO_AD_04_FLEXPWM1_PWM2_A** 0x400E811CU, 0x4U, 0x400E8508-U, 0x1U, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_GPIO_MUX3_IO03** 0x400E811CU, 0x5U, 0, 0, 0x400E8360-U
- #define **IOMUXC_GPIO_AD_04_WDOG1_B** 0x400E811CU, 0x6U, 0, 0, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_FLEXIO2_D04** 0x400E811CU, 0x8U, 0, 0, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_TMR4_TIMER0** 0x400E811CU, 0x9U, 0x400E8660U, 0x1U, 0x400E8360U
- #define **IOMUXC_GPIO_AD_04_GPIO9_IO03** 0x400E811CU, 0xAU, 0, 0, 0x400E8360U
- #define **IOMUXC_GPIO_AD_05_EMVSIM1_POWER_FAIL** 0x400E8120U, 0x0U, 0x400-E86A4U, 0x1U, 0x400E8364U
- #define **IOMUXC_GPIO_AD_05_LPUART8_RTS_B** 0x400E8120U, 0x1U, 0, 0, 0x400E8364U
- #define **IOMUXC_GPIO_AD_05_ENET_1G_1588_EVENT3_OUT** 0x400E8120U, 0x2U, 0, 0,

```

0x400E8364U
• #define IOMUXC_GPIO_AD_05_GPT2_CLK 0x400E8120U, 0x3U, 0, 0, 0x400E8364U
• #define IOMUXC_GPIO_AD_05_FLEXPWM1_PWM2_B 0x400E8120U, 0x4U, 0x400E8514-
  U, 0x1U, 0x400E8364U
• #define IOMUXC_GPIO_AD_05_GPIO_MUX3_IO04 0x400E8120U, 0x5U, 0, 0, 0x400E8364-
  U
• #define IOMUXC_GPIO_AD_05_WDOG2_B 0x400E8120U, 0x6U, 0, 0, 0x400E8364U
• #define IOMUXC_GPIO_AD_05_FLEXIO2_D05 0x400E8120U, 0x8U, 0, 0, 0x400E8364U
• #define IOMUXC_GPIO_AD_05_TMR4_TIMER1 0x400E8120U, 0x9U, 0x400E8664U, 0x1U,
  0x400E8364U
• #define IOMUXC_GPIO_AD_05_GPIO9_IO04 0x400E8120U, 0xAU, 0, 0, 0x400E8364U
• #define IOMUXC_GPIO_AD_06_USB_OTG2_OC 0x400E8124U, 0x0U, 0x400E86B8U, 0x0U,
  0x400E8368U
• #define IOMUXC_GPIO_AD_06_FLEXCAN1_TX 0x400E8124U, 0x1U, 0, 0, 0x400E8368U
• #define IOMUXC_GPIO_AD_06_EMVSIM2_IO 0x400E8124U, 0x2U, 0x400E86A8U, 0x0U,
  0x400E8368U
• #define IOMUXC_GPIO_AD_06_GPT3_CAPTURE1 0x400E8124U, 0x3U, 0x400E8590U,
  0x1U, 0x400E8368U
• #define IOMUXC_GPIO_AD_06_VIDEO_MUX_CSI_DATA15 0x400E8124U, 0x4U, 0, 0,
  0x400E8368U
• #define IOMUXC_GPIO_AD_06_GPIO_MUX3_IO05 0x400E8124U, 0x5U, 0, 0, 0x400E8368-
  U
• #define IOMUXC_GPIO_AD_06_ENET_1588_EVENT1_IN 0x400E8124U, 0x6U, 0, 0, 0x400-
  E8368U
• #define IOMUXC_GPIO_AD_06_FLEXIO2_D06 0x400E8124U, 0x8U, 0, 0, 0x400E8368U
• #define IOMUXC_GPIO_AD_06_TMR4_TIMER2 0x400E8124U, 0x9U, 0x400E8668U, 0x0U,
  0x400E8368U
• #define IOMUXC_GPIO_AD_06_GPIO9_IO05 0x400E8124U, 0xAU, 0, 0, 0x400E8368U
• #define IOMUXC_GPIO_AD_06_FLEXPWM1_PWM0_X 0x400E8124U, 0xBU, 0, 0, 0x400-
  E8368U
• #define IOMUXC_GPIO_AD_07_USB_OTG2_PWR 0x400E8128U, 0x0U, 0, 0, 0x400E836CU
• #define IOMUXC_GPIO_AD_07_FLEXCAN1_RX 0x400E8128U, 0x1U, 0x400E8498U, 0x0U,
  0x400E836CU
• #define IOMUXC_GPIO_AD_07_EMVSIM2_CLK 0x400E8128U, 0x2U, 0, 0, 0x400E836CU
• #define IOMUXC_GPIO_AD_07_GPT3_CAPTURE2 0x400E8128U, 0x3U, 0x400E8594U,
  0x1U, 0x400E836CU
• #define IOMUXC_GPIO_AD_07_VIDEO_MUX_CSI_DATA14 0x400E8128U, 0x4U, 0, 0,
  0x400E836CU
• #define IOMUXC_GPIO_AD_07_GPIO_MUX3_IO06 0x400E8128U, 0x5U, 0, 0, 0x400E836-
  CU
• #define IOMUXC_GPIO_AD_07_ENET_1588_EVENT1_OUT 0x400E8128U, 0x6U, 0, 0,
  0x400E836CU
• #define IOMUXC_GPIO_AD_07_FLEXIO2_D07 0x400E8128U, 0x8U, 0, 0, 0x400E836CU
• #define IOMUXC_GPIO_AD_07_TMR4_TIMER3 0x400E8128U, 0x9U, 0, 0, 0x400E836CU
• #define IOMUXC_GPIO_AD_07_GPIO9_IO06 0x400E8128U, 0xAU, 0, 0, 0x400E836CU
• #define IOMUXC_GPIO_AD_07_FLEXPWM1_PWM1_X 0x400E8128U, 0xBU, 0, 0, 0x400-
  E836CU
• #define IOMUXC_GPIO_AD_08_USBPHY2_OTG_ID 0x400E812CU, 0x0U, 0x400E86C4U,
  0x0U, 0x400E8370U
• #define IOMUXC_GPIO_AD_08_LPI2C1_SCL 0x400E812CU, 0x1U, 0x400E85ACU, 0x0U,
  0x400E8370U

```

- #define **IOMUXC_GPIO_AD_08_EMVSIM2_RST** 0x400E812CU, 0x2U, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_GPT3_COMPARE1** 0x400E812CU, 0x3U, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_VIDEO_MUX_CSI_DATA13** 0x400E812CU, 0x4U, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_GPIO_MUX3_IO07** 0x400E812CU, 0x5U, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_ENET_1588_EVENT2_IN** 0x400E812CU, 0x6U, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_FLEXIO2_D08** 0x400E812CU, 0x8U, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_GPIO9_IO07** 0x400E812CU, 0xAU, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_08_FLEXPWM1_PWM2_X** 0x400E812CU, 0xBU, 0, 0, 0x400E8370U
- #define **IOMUXC_GPIO_AD_09_USBPHY1_OTG_ID** 0x400E8130U, 0x0U, 0x400E86C0U, 0x0U, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_LPI2C1_SDA** 0x400E8130U, 0x1U, 0x400E85B0U, 0x0U, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_EMVSIM2_SVEN** 0x400E8130U, 0x2U, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_GPT3_COMPARE2** 0x400E8130U, 0x3U, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_VIDEO_MUX_CSI_DATA12** 0x400E8130U, 0x4U, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_GPIO_MUX3_IO08** 0x400E8130U, 0x5U, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_ENET_1588_EVENT2_OUT** 0x400E8130U, 0x6U, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_FLEXIO2_D09** 0x400E8130U, 0x8U, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_GPIO9_IO08** 0x400E8130U, 0xAU, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_09_FLEXPWM1_PWM3_X** 0x400E8130U, 0xBU, 0, 0, 0x400E8374U
- #define **IOMUXC_GPIO_AD_10_USB_OTG1_PWR** 0x400E8134U, 0x0U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_LPI2C1_SCL** 0x400E8134U, 0x1U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_EMVSIM2_PD** 0x400E8134U, 0x2U, 0x400E86ACU, 0x0U, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_GPT3_COMPARE3** 0x400E8134U, 0x3U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_VIDEO_MUX_CSI_DATA11** 0x400E8134U, 0x4U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_GPIO_MUX3_IO09** 0x400E8134U, 0x5U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_ENET_1588_EVENT3_IN** 0x400E8134U, 0x6U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_FLEXIO2_D10** 0x400E8134U, 0x8U, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_GPIO9_IO09** 0x400E8134U, 0xAU, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_10_FLEXPWM2_PWM0_X** 0x400E8134U, 0xBU, 0, 0, 0x400E8378U
- #define **IOMUXC_GPIO_AD_11_USB_OTG1_OC** 0x400E8138U, 0x0U, 0x400E86BCU, 0x0U, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_LPI2C1_SDAS** 0x400E8138U, 0x1U, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_EMVSIM2_POWER_FAIL** 0x400E8138U, 0x2U, 0x400E86B0U, 0x0U, 0x400E837CU

- #define **IOMUXC_GPIO_AD_11_GPT3_CLK** 0x400E8138U, 0x3U, 0x400E8598U, 0x1U, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_VIDEO_MUX_CSI_DATA10** 0x400E8138U, 0x4U, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_GPIO_MUX3_IO10** 0x400E8138U, 0x5U, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_ENET_1588_EVENT3_OUT** 0x400E8138U, 0x6U, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_FLEXIO2_D11** 0x400E8138U, 0x8U, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_GPIO9_IO10** 0x400E8138U, 0xAU, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_11_FLEXPWM2_PWM1_X** 0x400E8138U, 0xBU, 0, 0, 0x400E837CU
- #define **IOMUXC_GPIO_AD_12_SPDIF_LOCK** 0x400E813CU, 0x0U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_LPI2C1_HREQ** 0x400E813CU, 0x1U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_GPT1_CAPTURE1** 0x400E813CU, 0x2U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_FLEXSPI1_B_DATA03** 0x400E813CU, 0x3U, 0x400E8570U, 0x0U, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_VIDEO_MUX_CSI_PIXCLK** 0x400E813CU, 0x4U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_GPIO_MUX3_IO11** 0x400E813CU, 0x5U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_ENET_TX_DATA03** 0x400E813CU, 0x6U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_FLEXIO2_D12** 0x400E813CU, 0x8U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_EWM_OUT_B** 0x400E813CU, 0x9U, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_GPIO9_IO11** 0x400E813CU, 0xAU, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_12_FLEXPWM2_PWM2_X** 0x400E813CU, 0xBU, 0, 0, 0x400E8380U
- #define **IOMUXC_GPIO_AD_13_SPDIF_SR_CLK** 0x400E8140U, 0x0U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_PIT1_TRIGGER0** 0x400E8140U, 0x1U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_GPT1_CAPTURE2** 0x400E8140U, 0x2U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_FLEXSPI1_B_DATA02** 0x400E8140U, 0x3U, 0x400E856CU, 0x0U, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_VIDEO_MUX_CSI_MCLK** 0x400E8140U, 0x4U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_GPIO_MUX3_IO12** 0x400E8140U, 0x5U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_ENET_TX_DATA02** 0x400E8140U, 0x6U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_FLEXIO2_D13** 0x400E8140U, 0x8U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_REF_CLK_32K** 0x400E8140U, 0x9U, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_GPIO9_IO12** 0x400E8140U, 0xAU, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_13_FLEXPWM2_PWM3_X** 0x400E8140U, 0xBU, 0, 0, 0x400E8384U
- #define **IOMUXC_GPIO_AD_14_SPDIF_EXT_CLK** 0x400E8144U, 0x0U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_REF_CLK_24M** 0x400E8144U, 0x1U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_GPT1_COMPARE1** 0x400E8144U, 0x2U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_FLEXSPI1_B_DATA01** 0x400E8144U, 0x3U, 0x400E8568U, 0x0U, 0x400E8388U

- #define **IOMUXC_GPIO_AD_14_VIDEO_MUX_CSI_VSYNC** 0x400E8144U, 0x4U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_GPIO_MUX3_IO13** 0x400E8144U, 0x5U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_ENET_RX_CLK** 0x400E8144U, 0x6U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_FLEXIO2_D14** 0x400E8144U, 0x8U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_CCM_ENET_REF_CLK_25M** 0x400E8144U, 0x9U, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_GPIO9_IO13** 0x400E8144U, 0xAU, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_14_FLEXPWM3_PWM0_X** 0x400E8144U, 0xBU, 0, 0, 0x400E8388U
- #define **IOMUXC_GPIO_AD_15_GPIO9_IO14** 0x400E8148U, 0xAU, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_FLEXPWM3_PWM1_X** 0x400E8148U, 0xBU, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_SPDIF_IN** 0x400E8148U, 0x0U, 0x400E86B4U, 0x1U, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_LPUART10_TXD** 0x400E8148U, 0x1U, 0x400E8628U, 0x0U, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_GPT1_COMPARE2** 0x400E8148U, 0x2U, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_FLEXSPI1_B_DATA00** 0x400E8148U, 0x3U, 0x400E8564U, 0x0U, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_VIDEO_MUX_CSI_HSYNC** 0x400E8148U, 0x4U, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_GPIO_MUX3_IO14** 0x400E8148U, 0x5U, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_ENET_TX_ER** 0x400E8148U, 0x6U, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_15_FLEXIO2_D15** 0x400E8148U, 0x8U, 0, 0, 0x400E838CU
- #define **IOMUXC_GPIO_AD_16_SPDIF_OUT** 0x400E814CU, 0x0U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_LPUART10_RXD** 0x400E814CU, 0x1U, 0x400E8624U, 0x0U, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_GPT1_COMPARE3** 0x400E814CU, 0x2U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_FLEXSPI1_B_SCLK** 0x400E814CU, 0x3U, 0x400E8578U, 0x0U, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_VIDEO_MUX_CSI_DATA09** 0x400E814CU, 0x4U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_GPIO_MUX3_IO15** 0x400E814CU, 0x5U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_ENET_RX_DATA03** 0x400E814CU, 0x6U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_FLEXIO2_D16** 0x400E814CU, 0x8U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_ENET_1G_MDC** 0x400E814CU, 0x9U, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_GPIO9_IO15** 0x400E814CU, 0xAU, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_16_FLEXPWM3_PWM2_X** 0x400E814CU, 0xBU, 0, 0, 0x400E8390U
- #define **IOMUXC_GPIO_AD_17_SAI1_MCLK** 0x400E8150U, 0x0U, 0x400E866CU, 0x0U, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_ACMP1_OUT** 0x400E8150U, 0x1U, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_GPT1_CLK** 0x400E8150U, 0x2U, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_FLEXSPI1_A_DQS** 0x400E8150U, 0x3U, 0x400E8550U,

- 0x1U, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_VIDEO_MUX_CSI_DATA08** 0x400E8150U, 0x4U, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_GPIO_MUX3_IO16** 0x400E8150U, 0x5U, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_ENET_RX_DATA02** 0x400E8150U, 0x6U, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_FLEXIO2_D17** 0x400E8150U, 0x8U, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_ENET_1G_MDIO** 0x400E8150U, 0x9U, 0x400E84C8U, 0x2U, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_GPIO9_IO16** 0x400E8150U, 0xAU, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_17_FLEXPWM3_PWM3_X** 0x400E8150U, 0xBU, 0, 0, 0x400E8394U
- #define **IOMUXC_GPIO_AD_18_GPIO9_IO17** 0x400E8154U, 0xAU, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_FLEXPWM4_PWM0_X** 0x400E8154U, 0xBU, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_SAI1_RX_SYNC** 0x400E8154U, 0x0U, 0x400E8678U, 0x0U, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_ACMP2_OUT** 0x400E8154U, 0x1U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_LPSP1_PCS1** 0x400E8154U, 0x2U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_FLEXSPI1_A_SS0_B** 0x400E8154U, 0x3U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_VIDEO_MUX_CSI_DATA07** 0x400E8154U, 0x4U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_GPIO_MUX3_IO17** 0x400E8154U, 0x5U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_ENET_CRS** 0x400E8154U, 0x6U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_FLEXIO2_D18** 0x400E8154U, 0x8U, 0, 0, 0x400E8398U
- #define **IOMUXC_GPIO_AD_18_LPI2C2_SCL** 0x400E8154U, 0x9U, 0x400E85B4U, 0x1U, 0x400E8398U
- #define **IOMUXC_GPIO_AD_19_SAI1_RX_BCLK** 0x400E8158U, 0x0U, 0x400E8670U, 0x0U, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_ACMP3_OUT** 0x400E8158U, 0x1U, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_LPSP1_PCS2** 0x400E8158U, 0x2U, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_FLEXSPI1_A_SCLK** 0x400E8158U, 0x3U, 0x400E8574U, 0x0U, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_VIDEO_MUX_CSI_DATA06** 0x400E8158U, 0x4U, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_GPIO_MUX3_IO18** 0x400E8158U, 0x5U, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_ENET_COL** 0x400E8158U, 0x6U, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_FLEXIO2_D19** 0x400E8158U, 0x8U, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_LPI2C2_SDA** 0x400E8158U, 0x9U, 0x400E85B8U, 0x1U, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_GPIO9_IO18** 0x400E8158U, 0xAU, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_19_FLEXPWM4_PWM1_X** 0x400E8158U, 0xBU, 0, 0, 0x400E839CU
- #define **IOMUXC_GPIO_AD_20_SAI1_RX_DATA00** 0x400E815CU, 0x0U, 0x400E8674U, 0x0U, 0x400E83A0U
- #define **IOMUXC_GPIO_AD_20_ACMP4_OUT** 0x400E815CU, 0x1U, 0, 0, 0x400E83A0U
- #define **IOMUXC_GPIO_AD_20_LPSP1_PCS3** 0x400E815CU, 0x2U, 0, 0, 0x400E83A0U

- **#define IOMUXC_GPIO_AD_20_FLEXSPI1_A_DATA00** 0x400E815CU, 0x3U, 0x400E8554U, 0x0U, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_VIDEO_MUX_CSI_DATA05** 0x400E815CU, 0x4U, 0, 0, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_GPIO_MUX3_IO19** 0x400E815CU, 0x5U, 0, 0, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_KPP_ROW07** 0x400E815CU, 0x6U, 0x400E85A8U, 0x0U, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_FLEXIO2_D20** 0x400E815CU, 0x8U, 0, 0, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_ENET_QOS_1588_EVENT2_OUT** 0x400E815CU, 0x9U, 0, 0, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_GPIO9_IO19** 0x400E815CU, 0xAU, 0, 0, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_20_FLEXPWM4_PWM2_X** 0x400E815CU, 0xBU, 0, 0, 0x400E83A0U
- **#define IOMUXC_GPIO_AD_21_SAI1_TX_DATA00** 0x400E8160U, 0x0U, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_LPSP12_PCS1** 0x400E8160U, 0x2U, 0x400E85E0U, 0x0U, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_FLEXSPI1_A_DATA01** 0x400E8160U, 0x3U, 0x400E8558U, 0x0U, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_VIDEO_MUX_CSI_DATA04** 0x400E8160U, 0x4U, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_GPIO_MUX3_IO20** 0x400E8160U, 0x5U, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_KPP_COL07** 0x400E8160U, 0x6U, 0x400E85A0U, 0x0U, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_FLEXIO2_D21** 0x400E8160U, 0x8U, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_ENET_QOS_1588_EVENT2_IN** 0x400E8160U, 0x9U, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_GPIO9_IO20** 0x400E8160U, 0xAU, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_21_FLEXPWM4_PWM3_X** 0x400E8160U, 0xBU, 0, 0, 0x400E83A4U
- **#define IOMUXC_GPIO_AD_22_GPIO9_IO21** 0x400E8164U, 0xAU, 0, 0, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_SAI1_TX_BCLK** 0x400E8164U, 0x0U, 0x400E867CU, 0x0U, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_LPSP12_PCS2** 0x400E8164U, 0x2U, 0, 0, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_FLEXSPI1_A_DATA02** 0x400E8164U, 0x3U, 0x400E855CU, 0x0U, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_VIDEO_MUX_CSI_DATA03** 0x400E8164U, 0x4U, 0, 0, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_GPIO_MUX3_IO21** 0x400E8164U, 0x5U, 0, 0, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_KPP_ROW06** 0x400E8164U, 0x6U, 0x400E85A4U, 0x0U, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_FLEXIO2_D22** 0x400E8164U, 0x8U, 0, 0, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_22_ENET_QOS_1588_EVENT3_OUT** 0x400E8164U, 0x9U, 0, 0, 0x400E83A8U
- **#define IOMUXC_GPIO_AD_23_SAI1_TX_SYNC** 0x400E8168U, 0x0U, 0x400E8680U, 0x0U, 0x400E83ACU
- **#define IOMUXC_GPIO_AD_23_LPSP12_PCS3** 0x400E8168U, 0x2U, 0, 0, 0x400E83ACU
- **#define IOMUXC_GPIO_AD_23_FLEXSPI1_A_DATA03** 0x400E8168U, 0x3U, 0x400E8560U, 0x0U, 0x400E83ACU

- U, 0x0U, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_23_VIDEO_MUX_CSI_DATA02** 0x400E8168U, 0x4U, 0, 0, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_23_GPIO_MUX3_IO22** 0x400E8168U, 0x5U, 0, 0, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_23_KPP_COL06** 0x400E8168U, 0x6U, 0x400E859CU, 0x0U, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_23_FLEXIO2_D23** 0x400E8168U, 0x8U, 0, 0, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_23_ENET_QOS_1588_EVENT3_IN** 0x400E8168U, 0x9U, 0, 0, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_23_GPIO9_IO22** 0x400E8168U, 0xAU, 0, 0, 0x400E83ACU
- #define **IOMUXC_GPIO_AD_24_LPUART1_TXD** 0x400E816CU, 0x0U, 0x400E8620U, 0x0U, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_LPSPI2_SCK** 0x400E816CU, 0x1U, 0x400E85E4U, 0x0U, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_VIDEO_MUX_CSI_DATA00** 0x400E816CU, 0x2U, 0, 0, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_ENET_RX_EN** 0x400E816CU, 0x3U, 0x400E84B8U, 0x0U, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_FLEXPWM2_PWM0_A** 0x400E816CU, 0x4U, 0x400E8518U, 0x1U, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_GPIO_MUX3_IO23** 0x400E816CU, 0x5U, 0, 0, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_KPP_ROW05** 0x400E816CU, 0x6U, 0, 0, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_FLEXIO2_D24** 0x400E816CU, 0x8U, 0, 0, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_LPI2C4_SCL** 0x400E816CU, 0x9U, 0x400E85C4U, 0x0U, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_24_GPIO9_IO23** 0x400E816CU, 0xAU, 0, 0, 0x400E83B0U
- #define **IOMUXC_GPIO_AD_25_GPIO9_IO24** 0x400E8170U, 0xAU, 0, 0, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_LPUART1_RXD** 0x400E8170U, 0x0U, 0x400E861CU, 0x0U, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_LPSPI2_PCS0** 0x400E8170U, 0x1U, 0x400E85DCU, 0x0U, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_VIDEO_MUX_CSI_DATA01** 0x400E8170U, 0x2U, 0, 0, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_ENET_RX_ER** 0x400E8170U, 0x3U, 0x400E84BCU, 0x0U, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_FLEXPWM2_PWM0_B** 0x400E8170U, 0x4U, 0x400E8524U, 0x1U, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_GPIO_MUX3_IO24** 0x400E8170U, 0x5U, 0, 0, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_KPP_COL05** 0x400E8170U, 0x6U, 0, 0, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_FLEXIO2_D25** 0x400E8170U, 0x8U, 0, 0, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_25_LPI2C4_SDA** 0x400E8170U, 0x9U, 0x400E85C8U, 0x0U, 0x400E83B4U
- #define **IOMUXC_GPIO_AD_26_LPUART1_CTS_B** 0x400E8174U, 0x0U, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_LPSPI2_SOUT** 0x400E8174U, 0x1U, 0x400E85ECU, 0x0U, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_SEMC_CSX01** 0x400E8174U, 0x2U, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_ENET_RX_DATA00** 0x400E8174U, 0x3U, 0x400E84B0U, 0x0U, 0x400E83B8U

- #define **IOMUXC_GPIO_AD_26_FLEXPWM2_PWM1_A** 0x400E8174U, 0x4U, 0x400E851CU, 0x1U, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_GPIO_MUX3_IO25** 0x400E8174U, 0x5U, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_KPP_ROW04** 0x400E8174U, 0x6U, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_FLEXIO2_D26** 0x400E8174U, 0x8U, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_ENET_QOS_MDC** 0x400E8174U, 0x9U, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_GPIO9_IO25** 0x400E8174U, 0xAU, 0, 0, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_26_USDHC2_CD_B** 0x400E8174U, 0xBU, 0x400E86D0U, 0x1U, 0x400E83B8U
- #define **IOMUXC_GPIO_AD_27_LPUART1_RTS_B** 0x400E8178U, 0x0U, 0, 0, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_LPSPI2_SIN** 0x400E8178U, 0x1U, 0x400E85E8U, 0x0U, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_SEMC_CSX02** 0x400E8178U, 0x2U, 0, 0, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_ENET_RX_DATA01** 0x400E8178U, 0x3U, 0x400E84B4U, 0x0U, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_FLEXPWM2_PWM1_B** 0x400E8178U, 0x4U, 0x400E8528U, 0x1U, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_GPIO_MUX3_IO26** 0x400E8178U, 0x5U, 0, 0, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_KPP_COL04** 0x400E8178U, 0x6U, 0, 0, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_FLEXIO2_D27** 0x400E8178U, 0x8U, 0, 0, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_ENET_QOS_MDIO** 0x400E8178U, 0x9U, 0x400E84ECU, 0x1U, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_GPIO9_IO26** 0x400E8178U, 0xAU, 0, 0, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_27_USDHC2_WP** 0x400E8178U, 0xBU, 0x400E86D4U, 0x1U, 0x400E83BCU
- #define **IOMUXC_GPIO_AD_28_GPIO9_IO27** 0x400E817CU, 0xAU, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_USDHC2_VSELECT** 0x400E817CU, 0xBU, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_LPSPI1_SCK** 0x400E817CU, 0x0U, 0x400E85D0U, 0x1U, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_LPUART5_TXD** 0x400E817CU, 0x1U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_SEMC_CSX03** 0x400E817CU, 0x2U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_ENET_TX_EN** 0x400E817CU, 0x3U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_FLEXPWM2_PWM2_A** 0x400E817CU, 0x4U, 0x400E8520U, 0x1U, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_GPIO_MUX3_IO27** 0x400E817CU, 0x5U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_KPP_ROW03** 0x400E817CU, 0x6U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_FLEXIO2_D28** 0x400E817CU, 0x8U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_28_VIDEO_MUX_EXT_DCIC1** 0x400E817CU, 0x9U, 0, 0, 0x400E83C0U
- #define **IOMUXC_GPIO_AD_29_LPSPI1_PCS0** 0x400E8180U, 0x0U, 0x400E85CCU, 0x1U, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_LPUART5_RXD** 0x400E8180U, 0x1U, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_ENET_REF_CLK** 0x400E8180U, 0x2U, 0x400E84A8U, 0x0U, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_ENET_TX_CLK** 0x400E8180U, 0x3U, 0x400E84C0U, 0x0U, 0x400E83C4U

- #define **IOMUXC_GPIO_AD_29_FLEXPWM2_PWM2_B** 0x400E8180U, 0x4U, 0x400E852CU, 0x1U, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_GPIO_MUX3_IO28** 0x400E8180U, 0x5U, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_KPP_COL03** 0x400E8180U, 0x6U, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_FLEXIO2_D29** 0x400E8180U, 0x8U, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_VIDEO_MUX_EXT_DCIC2** 0x400E8180U, 0x9U, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_GPIO9_IO28** 0x400E8180U, 0xAU, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_29_USDHC2_RESET_B** 0x400E8180U, 0xBU, 0, 0, 0x400E83C4U
- #define **IOMUXC_GPIO_AD_30_LPSPI1_SOUT** 0x400E8184U, 0x0U, 0x400E85D8U, 0x1U, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_USB_OTG2_OC** 0x400E8184U, 0x1U, 0x400E86B8U, 0x1U, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_FLEXCAN2_TX** 0x400E8184U, 0x2U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_ENET_TX_DATA00** 0x400E8184U, 0x3U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_LPUART3_TXD** 0x400E8184U, 0x4U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_GPIO_MUX3_IO29** 0x400E8184U, 0x5U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_KPP_ROW02** 0x400E8184U, 0x6U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_FLEXIO2_D30** 0x400E8184U, 0x8U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_WDOG2_RESET_B_DEB** 0x400E8184U, 0x9U, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_30_GPIO9_IO29** 0x400E8184U, 0xAU, 0, 0, 0x400E83C8U
- #define **IOMUXC_GPIO_AD_31_LPSPI1_SIN** 0x400E8188U, 0x0U, 0x400E85D4U, 0x1U, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_USB_OTG2_PWR** 0x400E8188U, 0x1U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_FLEXCAN2_RX** 0x400E8188U, 0x2U, 0x400E849CU, 0x1U, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_ENET_TX_DATA01** 0x400E8188U, 0x3U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_LPUART3_RXD** 0x400E8188U, 0x4U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_GPIO_MUX3_IO30** 0x400E8188U, 0x5U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_KPP_COL02** 0x400E8188U, 0x6U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_FLEXIO2_D31** 0x400E8188U, 0x8U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_WDOG1_RESET_B_DEB** 0x400E8188U, 0x9U, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_31_GPIO9_IO30** 0x400E8188U, 0xAU, 0, 0, 0x400E83CCU
- #define **IOMUXC_GPIO_AD_32_GPIO9_IO31** 0x400E818CU, 0xAU, 0, 0, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_LPI2C1_SCL** 0x400E818CU, 0x0U, 0x400E85ACU, 0x1U, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_USBPHY2_OTG_ID** 0x400E818CU, 0x1U, 0x400E86C4U, 0x1U, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_PGMC_PMIC_RDY** 0x400E818CU, 0x2U, 0, 0, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_ENET_MDC** 0x400E818CU, 0x3U, 0, 0, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_USDHC1_CD_B** 0x400E818CU, 0x4U, 0x400E86C8U, 0x0U, 0x400E83D0U

- #define **IOMUXC_GPIO_AD_32_GPIO_MUX3_IO31** 0x400E818CU, 0x5U, 0, 0, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_KPP_ROW01** 0x400E818CU, 0x6U, 0, 0, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_LPUART10_TXD** 0x400E818CU, 0x8U, 0x400E8628U, 0x1U, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_32_ENET_1G_MDC** 0x400E818CU, 0x9U, 0, 0, 0x400E83D0U
- #define **IOMUXC_GPIO_AD_33_LPI2C1_SDA** 0x400E8190U, 0x0U, 0x400E85B0U, 0x1U, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_USBPHY1_OTG_ID** 0x400E8190U, 0x1U, 0x400E86C0U, 0x1U, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_XBAR1_INOUT17** 0x400E8190U, 0x2U, 0, 0, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_ENET_MDIO** 0x400E8190U, 0x3U, 0x400E84ACU, 0x1U, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_USDHC1_WP** 0x400E8190U, 0x4U, 0x400E86CCU, 0x0U, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_GPIO_MUX4_IO00** 0x400E8190U, 0x5U, 0, 0, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_KPP_COL01** 0x400E8190U, 0x6U, 0, 0, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_LPUART10_RXD** 0x400E8190U, 0x8U, 0x400E8624U, 0x1U, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_ENET_1G_MDIO** 0x400E8190U, 0x9U, 0x400E84C8U, 0x3U, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_33_GPIO10_IO00** 0x400E8190U, 0xAU, 0, 0, 0x400E83D4U
- #define **IOMUXC_GPIO_AD_34_ENET_1G_1588_EVENT0_IN** 0x400E8194U, 0x0U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_USB_OTG1_PWR** 0x400E8194U, 0x1U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_XBAR1_INOUT18** 0x400E8194U, 0x2U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_ENET_1588_EVENT0_IN** 0x400E8194U, 0x3U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_USDHC1_VSELECT** 0x400E8194U, 0x4U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_GPIO_MUX4_IO01** 0x400E8194U, 0x5U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_KPP_ROW00** 0x400E8194U, 0x6U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_LPUART10_CTS_B** 0x400E8194U, 0x8U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_WDOG1_ANY** 0x400E8194U, 0x9U, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_34_GPIO10_IO01** 0x400E8194U, 0xAU, 0, 0, 0x400E83D8U
- #define **IOMUXC_GPIO_AD_35_GPIO10_IO02** 0x400E8198U, 0xAU, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_ENET_1G_1588_EVENT0_OUT** 0x400E8198U, 0x0U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_USB_OTG1_OC** 0x400E8198U, 0x1U, 0x400E86BCU, 0x1U, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_XBAR1_INOUT19** 0x400E8198U, 0x2U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_ENET_1588_EVENT0_OUT** 0x400E8198U, 0x3U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_USDHC1_RESET_B** 0x400E8198U, 0x4U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_GPIO_MUX4_IO02** 0x400E8198U, 0x5U, 0, 0, 0x400E83DCU

- #define **IOMUXC_GPIO_AD_35_KPP_COL00** 0x400E8198U, 0x6U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_LPUART10_RTS_B** 0x400E8198U, 0x8U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_AD_35_FLEXSPI1_B_SS1_B** 0x400E8198U, 0x9U, 0, 0, 0x400E83DCU
- #define **IOMUXC_GPIO_SD_B1_00_USDHC1_CMD** 0x400E819CU, 0x0U, 0, 0, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_00_XBAR1_INOUT20** 0x400E819CU, 0x2U, 0x400E86D8U, 0x1U, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_00_GPT4_CAPTURE1** 0x400E819CU, 0x3U, 0, 0, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_00_GPIO_MUX4_IO03** 0x400E819CU, 0x5U, 0, 0, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_00_FLEXSPI2_A_SS0_B** 0x400E819CU, 0x6U, 0, 0, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_00_KPP_ROW07** 0x400E819CU, 0x8U, 0x400E85A8U, 0x1U, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_00_GPIO10_IO03** 0x400E819CU, 0xAU, 0, 0, 0x400E83E0U
- #define **IOMUXC_GPIO_SD_B1_01_USDHC1_CLK** 0x400E81A0U, 0x0U, 0, 0, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_01_XBAR1_INOUT21** 0x400E81A0U, 0x2U, 0x400E86DCU, 0x1U, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_01_GPT4_CAPTURE2** 0x400E81A0U, 0x3U, 0, 0, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_01_GPIO_MUX4_IO04** 0x400E81A0U, 0x5U, 0, 0, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_01_FLEXSPI2_A_SCLK** 0x400E81A0U, 0x6U, 0x400E858CU, 0x1U, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_01_KPP_COL07** 0x400E81A0U, 0x8U, 0x400E85A0U, 0x1U, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_01_GPIO10_IO04** 0x400E81A0U, 0xAU, 0, 0, 0x400E83E4U
- #define **IOMUXC_GPIO_SD_B1_02_GPIO10_IO05** 0x400E81A4U, 0xAU, 0, 0, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_USDHC1_DATA0** 0x400E81A4U, 0x0U, 0, 0, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_XBAR1_INOUT22** 0x400E81A4U, 0x2U, 0x400E86E0U, 0x1U, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_GPT4_COMPARE1** 0x400E81A4U, 0x3U, 0, 0, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_GPIO_MUX4_IO05** 0x400E81A4U, 0x5U, 0, 0, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_FLEXSPI2_A_DATA00** 0x400E81A4U, 0x6U, 0x400E857CU, 0x1U, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_KPP_ROW06** 0x400E81A4U, 0x8U, 0x400E85A4U, 0x1U, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_02_FLEXSPI1_A_SS1_B** 0x400E81A4U, 0x9U, 0, 0, 0x400E83E8U
- #define **IOMUXC_GPIO_SD_B1_03_USDHC1_DATA1** 0x400E81A8U, 0x0U, 0, 0, 0x400E83ECU
- #define **IOMUXC_GPIO_SD_B1_03_XBAR1_INOUT23** 0x400E81A8U, 0x2U, 0x400E86E4U, 0x1U, 0x400E83ECU

- #define **IOMUXC_GPIO_SD_B1_03_GPT4_COMPARE2** 0x400E81A8U, 0x3U, 0, 0, 0x400-E83ECU
- #define **IOMUXC_GPIO_SD_B1_03_GPIO_MUX4_IO06** 0x400E81A8U, 0x5U, 0, 0, 0x400-E83ECU
- #define **IOMUXC_GPIO_SD_B1_03_FLEXSPI2_A_DATA01** 0x400E81A8U, 0x6U, 0x400-E8580U, 0x1U, 0x400E83ECU
- #define **IOMUXC_GPIO_SD_B1_03_KPP_COL06** 0x400E81A8U, 0x8U, 0x400E859CU, 0x1-U, 0x400E83ECU
- #define **IOMUXC_GPIO_SD_B1_03_FLEXSPI1_B_SS1_B** 0x400E81A8U, 0x9U, 0, 0, 0x400-E83ECU
- #define **IOMUXC_GPIO_SD_B1_03_GPIO10_IO06** 0x400E81A8U, 0xAU, 0, 0, 0x400E83EC-U
- #define **IOMUXC_GPIO_SD_B1_04_USDHC1_DATA2** 0x400E81ACU, 0x0U, 0, 0, 0x400E83-F0U
- #define **IOMUXC_GPIO_SD_B1_04_XBAR1_INOUT24** 0x400E81ACU, 0x2U, 0x400E86E8-U, 0x1U, 0x400E83F0U
- #define **IOMUXC_GPIO_SD_B1_04_GPT4_COMPARE3** 0x400E81ACU, 0x3U, 0, 0, 0x400-E83F0U
- #define **IOMUXC_GPIO_SD_B1_04_GPIO_MUX4_IO07** 0x400E81ACU, 0x5U, 0, 0, 0x400-E83F0U
- #define **IOMUXC_GPIO_SD_B1_04_FLEXSPI2_A_DATA02** 0x400E81ACU, 0x6U, 0x400-E8584U, 0x1U, 0x400E83F0U
- #define **IOMUXC_GPIO_SD_B1_04_FLEXSPI1_B_SS0_B** 0x400E81ACU, 0x8U, 0, 0, 0x400-E83F0U
- #define **IOMUXC_GPIO_SD_B1_04_ENET_QOS_1588_EVENT2_AUX_IN** 0x400E81ACU, 0x9U, 0, 0, 0x400E83F0U
- #define **IOMUXC_GPIO_SD_B1_04_GPIO10_IO07** 0x400E81ACU, 0xAU, 0, 0, 0x400E83F0U
- #define **IOMUXC_GPIO_SD_B1_05_GPIO10_IO08** 0x400E81B0U, 0xAU, 0, 0, 0x400E83F4U
- #define **IOMUXC_GPIO_SD_B1_05_USDHC1_DATA3** 0x400E81B0U, 0x0U, 0, 0, 0x400E83-F4U
- #define **IOMUXC_GPIO_SD_B1_05_XBAR1_INOUT25** 0x400E81B0U, 0x2U, 0x400E86ECU, 0x1U, 0x400E83F4U
- #define **IOMUXC_GPIO_SD_B1_05_GPT4_CLK** 0x400E81B0U, 0x3U, 0, 0, 0x400E83F4U
- #define **IOMUXC_GPIO_SD_B1_05_GPIO_MUX4_IO08** 0x400E81B0U, 0x5U, 0, 0, 0x400-E83F4U
- #define **IOMUXC_GPIO_SD_B1_05_FLEXSPI2_A_DATA03** 0x400E81B0U, 0x6U, 0x400-E8588U, 0x1U, 0x400E83F4U
- #define **IOMUXC_GPIO_SD_B1_05_FLEXSPI1_B_DQS** 0x400E81B0U, 0x8U, 0, 0, 0x400-E83F4U
- #define **IOMUXC_GPIO_SD_B1_05_ENET_QOS_1588_EVENT3_AUX_IN** 0x400E81B0U, 0x9U, 0, 0, 0x400E83F4U
- #define **IOMUXC_GPIO_SD_B2_00_GPIO10_IO09** 0x400E81B4U, 0xAU, 0, 0, 0x400E83F8U
- #define **IOMUXC_GPIO_SD_B2_00_USDHC2_DATA3** 0x400E81B4U, 0x0U, 0, 0, 0x400E83-F8U
- #define **IOMUXC_GPIO_SD_B2_00_FLEXSPI1_B_DATA03** 0x400E81B4U, 0x1U, 0x400-E8570U, 0x1U, 0x400E83F8U
- #define **IOMUXC_GPIO_SD_B2_00_ENET_1G_RX_EN** 0x400E81B4U, 0x2U, 0x400E84E0-U, 0x1U, 0x400E83F8U
- #define **IOMUXC_GPIO_SD_B2_00_LPUART9_TXD** 0x400E81B4U, 0x3U, 0, 0, 0x400E83-F8U

- #define **IOMUXC_GPIO_SD_B2_00_LPSPI4_SCK** 0x400E81B4U, 0x4U, 0x400E8610U, 0x0-U, 0x400E83F8U
- #define **IOMUXC_GPIO_SD_B2_00_GPIO_MUX4_IO09** 0x400E81B4U, 0x5U, 0, 0, 0x400-E83F8U
- #define **IOMUXC_GPIO_SD_B2_01_USDHC2_DATA2** 0x400E81B8U, 0x0U, 0, 0, 0x400E83-FCU
- #define **IOMUXC_GPIO_SD_B2_01_FLEXSPI1_B_DATA02** 0x400E81B8U, 0x1U, 0x400-E856CU, 0x1U, 0x400E83FCU
- #define **IOMUXC_GPIO_SD_B2_01_ENET_1G_RX_CLK** 0x400E81B8U, 0x2U, 0x400E84C-CU, 0x1U, 0x400E83FCU
- #define **IOMUXC_GPIO_SD_B2_01_LPUART9_RXD** 0x400E81B8U, 0x3U, 0, 0, 0x400E83F-CU
- #define **IOMUXC_GPIO_SD_B2_01_LPSPI4_PCS0** 0x400E81B8U, 0x4U, 0x400E860CU, 0x0-U, 0x400E83FCU
- #define **IOMUXC_GPIO_SD_B2_01_GPIO_MUX4_IO10** 0x400E81B8U, 0x5U, 0, 0, 0x400-E83FCU
- #define **IOMUXC_GPIO_SD_B2_01_GPIO10_IO10** 0x400E81B8U, 0xAU, 0, 0, 0x400E83FCU
- #define **IOMUXC_GPIO_SD_B2_02_GPIO10_IO11** 0x400E81BCU, 0xAU, 0, 0, 0x400E8400U
- #define **IOMUXC_GPIO_SD_B2_02_USDHC2_DATA1** 0x400E81BCU, 0x0U, 0, 0, 0x400-E8400U
- #define **IOMUXC_GPIO_SD_B2_02_FLEXSPI1_B_DATA01** 0x400E81BCU, 0x1U, 0x400-E8568U, 0x1U, 0x400E8400U
- #define **IOMUXC_GPIO_SD_B2_02_ENET_1G_RX_DATA00** 0x400E81BCU, 0x2U, 0x400-E84D0U, 0x1U, 0x400E8400U
- #define **IOMUXC_GPIO_SD_B2_02_LPUART9_CTS_B** 0x400E81BCU, 0x3U, 0, 0, 0x400-E8400U
- #define **IOMUXC_GPIO_SD_B2_02_LPSPI4_SOUT** 0x400E81BCU, 0x4U, 0x400E8618U, 0x0U, 0x400E8400U
- #define **IOMUXC_GPIO_SD_B2_02_GPIO_MUX4_IO11** 0x400E81BCU, 0x5U, 0, 0, 0x400-E8400U
- #define **IOMUXC_GPIO_SD_B2_03_GPIO10_IO12** 0x400E81C0U, 0xAU, 0, 0, 0x400E8404U
- #define **IOMUXC_GPIO_SD_B2_03_USDHC2_DATA0** 0x400E81C0U, 0x0U, 0, 0, 0x400-E8404U
- #define **IOMUXC_GPIO_SD_B2_03_FLEXSPI1_B_DATA00** 0x400E81C0U, 0x1U, 0x400-E8564U, 0x1U, 0x400E8404U
- #define **IOMUXC_GPIO_SD_B2_03_ENET_1G_RX_DATA01** 0x400E81C0U, 0x2U, 0x400-E84D4U, 0x1U, 0x400E8404U
- #define **IOMUXC_GPIO_SD_B2_03_LPUART9_RTS_B** 0x400E81C0U, 0x3U, 0, 0, 0x400-E8404U
- #define **IOMUXC_GPIO_SD_B2_03_LPSPI4_SIN** 0x400E81C0U, 0x4U, 0x400E8614U, 0x0U, 0x400E8404U
- #define **IOMUXC_GPIO_SD_B2_03_GPIO_MUX4_IO12** 0x400E81C0U, 0x5U, 0, 0, 0x400-E8404U
- #define **IOMUXC_GPIO_SD_B2_04_USDHC2_CLK** 0x400E81C4U, 0x0U, 0, 0, 0x400E8408U
- #define **IOMUXC_GPIO_SD_B2_04_FLEXSPI1_B_SCLK** 0x400E81C4U, 0x1U, 0x400-E8578U, 0x1U, 0x400E8408U
- #define **IOMUXC_GPIO_SD_B2_04_ENET_1G_RX_DATA02** 0x400E81C4U, 0x2U, 0x400-E84D8U, 0x1U, 0x400E8408U
- #define **IOMUXC_GPIO_SD_B2_04_FLEXSPI1_A_SS1_B** 0x400E81C4U, 0x3U, 0, 0, 0x400-E8408U

- #define **IOMUXC_GPIO_SD_B2_04_LPSPI4_PCS1** 0x400E81C4U, 0x4U, 0, 0, 0x400E8408U
- #define **IOMUXC_GPIO_SD_B2_04_GPIO_MUX4_IO13** 0x400E81C4U, 0x5U, 0, 0, 0x400E8408U
- #define **IOMUXC_GPIO_SD_B2_04_GPIO10_IO13** 0x400E81C4U, 0xAU, 0, 0, 0x400E8408U
- #define **IOMUXC_GPIO_SD_B2_05_GPIO10_IO14** 0x400E81C8U, 0xAU, 0, 0, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_05_USDHC2_CMD** 0x400E81C8U, 0x0U, 0, 0, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_05_FLEXSPI1_A_DQS** 0x400E81C8U, 0x1U, 0x400E8550U, 0x2U, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_05_ENET_1G_RX_DATA03** 0x400E81C8U, 0x2U, 0x400E84DCU, 0x1U, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_05_FLEXSPI1_B_SS0_B** 0x400E81C8U, 0x3U, 0, 0, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_05_LPSPI4_PCS2** 0x400E81C8U, 0x4U, 0, 0, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_05_GPIO_MUX4_IO14** 0x400E81C8U, 0x5U, 0, 0, 0x400E840CU
- #define **IOMUXC_GPIO_SD_B2_06_GPIO10_IO15** 0x400E81CCU, 0xAU, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_06_USDHC2_RESET_B** 0x400E81CCU, 0x0U, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_06_FLEXSPI1_A_SS0_B** 0x400E81CCU, 0x1U, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_06_ENET_1G_TX_DATA03** 0x400E81CCU, 0x2U, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_06_LPSPI4_PCS3** 0x400E81CCU, 0x3U, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_06_GPT6_CAPTURE1** 0x400E81CCU, 0x4U, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_06_GPIO_MUX4_IO15** 0x400E81CCU, 0x5U, 0, 0, 0x400E8410U
- #define **IOMUXC_GPIO_SD_B2_07_USDHC2_STROBE** 0x400E81D0U, 0x0U, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_FLEXSPI1_A_SCLK** 0x400E81D0U, 0x1U, 0x400E8574U, 0x1U, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_ENET_1G_TX_DATA02** 0x400E81D0U, 0x2U, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_LPUART3_CTS_B** 0x400E81D0U, 0x3U, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_GPT6_CAPTURE2** 0x400E81D0U, 0x4U, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_GPIO_MUX4_IO16** 0x400E81D0U, 0x5U, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_LPSPI2_SCK** 0x400E81D0U, 0x6U, 0x400E85E4U, 0x1U, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_ENET_TX_ER** 0x400E81D0U, 0x8U, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_ENET_QOS_REF_CLK** 0x400E81D0U, 0x9U, 0x400E84A0U, 0x1U, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_07_GPIO10_IO16** 0x400E81D0U, 0xAU, 0, 0, 0x400E8414U
- #define **IOMUXC_GPIO_SD_B2_08_GPIO10_IO17** 0x400E81D4U, 0xAU, 0, 0, 0x400E8418U
- #define **IOMUXC_GPIO_SD_B2_08_USDHC2_DATA4** 0x400E81D4U, 0x0U, 0, 0, 0x400E8418U
- #define **IOMUXC_GPIO_SD_B2_08_FLEXSPI1_A_DATA00** 0x400E81D4U, 0x1U, 0x400E8554U, 0x1U, 0x400E8418U

- **#define IOMUXC_GPIO_SD_B2_08_ENET_1G_TX_DATA01** 0x400E81D4U, 0x2U, 0, 0, 0x400E8418U
- **#define IOMUXC_GPIO_SD_B2_08_LPUART3_RTS_B** 0x400E81D4U, 0x3U, 0, 0, 0x400E8418U
- **#define IOMUXC_GPIO_SD_B2_08_GPT6_COMPARE1** 0x400E81D4U, 0x4U, 0, 0, 0x400E8418U
- **#define IOMUXC_GPIO_SD_B2_08_GPIO_MUX4_IO17** 0x400E81D4U, 0x5U, 0, 0, 0x400E8418U
- **#define IOMUXC_GPIO_SD_B2_08_LPSPI2_PCS0** 0x400E81D4U, 0x6U, 0x400E85DCU, 0x1U, 0x400E8418U
- **#define IOMUXC_GPIO_SD_B2_09_GPIO10_IO18** 0x400E81D8U, 0xAU, 0, 0, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_USDHC2_DATA5** 0x400E81D8U, 0x0U, 0, 0, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_FLEXSPI1_A_DATA01** 0x400E81D8U, 0x1U, 0x400E8558U, 0x1U, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_ENET_1G_TX_DATA00** 0x400E81D8U, 0x2U, 0, 0, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_LPUART5_CTS_B** 0x400E81D8U, 0x3U, 0, 0, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_GPT6_COMPARE2** 0x400E81D8U, 0x4U, 0, 0, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_GPIO_MUX4_IO18** 0x400E81D8U, 0x5U, 0, 0, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_09_LPSPI2_SOUT** 0x400E81D8U, 0x6U, 0x400E85ECU, 0x1U, 0x400E841CU
- **#define IOMUXC_GPIO_SD_B2_10_GPIO10_IO19** 0x400E81DCU, 0xAU, 0, 0, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_USDHC2_DATA6** 0x400E81DCU, 0x0U, 0, 0, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_FLEXSPI1_A_DATA02** 0x400E81DCU, 0x1U, 0x400E855CU, 0x1U, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_ENET_1G_TX_EN** 0x400E81DCU, 0x2U, 0, 0, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_LPUART5_RTS_B** 0x400E81DCU, 0x3U, 0, 0, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_GPT6_COMPARE3** 0x400E81DCU, 0x4U, 0, 0, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_GPIO_MUX4_IO19** 0x400E81DCU, 0x5U, 0, 0, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_10_LPSPI2_SIN** 0x400E81DCU, 0x6U, 0x400E85E8U, 0x1U, 0x400E8420U
- **#define IOMUXC_GPIO_SD_B2_11_USDHC2_DATA7** 0x400E81E0U, 0x0U, 0, 0, 0x400E8424U
- **#define IOMUXC_GPIO_SD_B2_11_FLEXSPI1_A_DATA03** 0x400E81E0U, 0x1U, 0x400E8560U, 0x1U, 0x400E8424U
- **#define IOMUXC_GPIO_SD_B2_11_ENET_1G_TX_CLK_IO** 0x400E81E0U, 0x2U, 0x400E84E8U, 0x1U, 0x400E8424U
- **#define IOMUXC_GPIO_SD_B2_11_ENET_1G_REF_CLK** 0x400E81E0U, 0x3U, 0x400E84C4U, 0x1U, 0x400E8424U
- **#define IOMUXC_GPIO_SD_B2_11_GPT6_CLK** 0x400E81E0U, 0x4U, 0, 0, 0x400E8424U
- **#define IOMUXC_GPIO_SD_B2_11_GPIO_MUX4_IO20** 0x400E81E0U, 0x5U, 0, 0, 0x400E8424U

- E8424U
- #define **IOMUXC_GPIO_SD_B2_11_LPSPI2_PCS1** 0x400E81E0U, 0x6U, 0x400E85E0U, 0x1-U, 0x400E8424U
- #define **IOMUXC_GPIO_SD_B2_11_GPIO10_IO20** 0x400E81E0U, 0xAU, 0, 0, 0x400E8424U
- #define **IOMUXC_GPIO_DISP_B1_00_VIDEO_MUX_LCDIF_CLK** 0x400E81E4U, 0x0U, 0, 0, 0x400E8428U
- #define **IOMUXC_GPIO_DISP_B1_00_ENET_1G_RX_EN** 0x400E81E4U, 0x1U, 0x400E84-E0U, 0x2U, 0x400E8428U
- #define **IOMUXC_GPIO_DISP_B1_00_TMR1_TIMER0** 0x400E81E4U, 0x3U, 0x400E863CU, 0x2U, 0x400E8428U
- #define **IOMUXC_GPIO_DISP_B1_00_XBAR1_INOUT26** 0x400E81E4U, 0x4U, 0x400E86-F0U, 0x1U, 0x400E8428U
- #define **IOMUXC_GPIO_DISP_B1_00_GPIO_MUX4_IO21** 0x400E81E4U, 0x5U, 0, 0, 0x400-E8428U
- #define **IOMUXC_GPIO_DISP_B1_00_ENET_QOS_RX_EN** 0x400E81E4U, 0x8U, 0x400-E84F8U, 0x0U, 0x400E8428U
- #define **IOMUXC_GPIO_DISP_B1_00_GPIO10_IO21** 0x400E81E4U, 0xAU, 0, 0, 0x400-E8428U
- #define **IOMUXC_GPIO_DISP_B1_01_VIDEO_MUX_LCDIF_ENABLE** 0x400E81E8U, 0x0-U, 0, 0, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_ENET_1G_RX_CLK** 0x400E81E8U, 0x1U, 0x400E84-CCU, 0x2U, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_ENET_1G_RX_ER** 0x400E81E8U, 0x2U, 0x400E84-E4U, 0x1U, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_TMR1_TIMER1** 0x400E81E8U, 0x3U, 0x400E8640U, 0x2U, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_XBAR1_INOUT27** 0x400E81E8U, 0x4U, 0x400E86-F4U, 0x1U, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_GPIO_MUX4_IO22** 0x400E81E8U, 0x5U, 0, 0, 0x400-E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_ENET_QOS_RX_CLK** 0x400E81E8U, 0x8U, 0, 0, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_ENET_QOS_RX_ER** 0x400E81E8U, 0x9U, 0x400-E84FCU, 0x0U, 0x400E842CU
- #define **IOMUXC_GPIO_DISP_B1_01_GPIO10_IO22** 0x400E81E8U, 0xAU, 0, 0, 0x400E842-CU
- #define **IOMUXC_GPIO_DISP_B1_02_GPIO10_IO23** 0x400E81ECU, 0xAU, 0, 0, 0x400-E8430U
- #define **IOMUXC_GPIO_DISP_B1_02_VIDEO_MUX_LCDIF_HSYNC** 0x400E81ECU, 0x0-U, 0, 0, 0x400E8430U
- #define **IOMUXC_GPIO_DISP_B1_02_ENET_1G_RX_DATA00** 0x400E81ECU, 0x1U, 0x400E84D0U, 0x2U, 0x400E8430U
- #define **IOMUXC_GPIO_DISP_B1_02_LPI2C3_SCL** 0x400E81ECU, 0x2U, 0x400E85BCU, 0x0U, 0x400E8430U
- #define **IOMUXC_GPIO_DISP_B1_02_TMR1_TIMER2** 0x400E81ECU, 0x3U, 0x400E8644U, 0x1U, 0x400E8430U
- #define **IOMUXC_GPIO_DISP_B1_02_XBAR1_INOUT28** 0x400E81ECU, 0x4U, 0x400E86-F8U, 0x1U, 0x400E8430U
- #define **IOMUXC_GPIO_DISP_B1_02_GPIO_MUX4_IO23** 0x400E81ECU, 0x5U, 0, 0, 0x400-E8430U

- **#define IOMUXC_GPIO_DISP_B1_02_ENET_QOS_RX_DATA00** 0x400E81ECU, 0x8U, 0x400E84F0U, 0x0U, 0x400E8430U
- **#define IOMUXC_GPIO_DISP_B1_02_LPUART1_TXD** 0x400E81ECU, 0x9U, 0x400E8620U, 0x1U, 0x400E8430U
- **#define IOMUXC_GPIO_DISP_B1_03_VIDEO_MUX_LCDIF_VSYNC** 0x400E81F0U, 0x0U, 0, 0, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_ENET_1G_RX_DATA01** 0x400E81F0U, 0x1U, 0x400E84D4U, 0x2U, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_LPI2C3_SDA** 0x400E81F0U, 0x2U, 0x400E85C0U, 0x0U, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_TMR2_TIMER0** 0x400E81F0U, 0x3U, 0x400E8648U, 0x2U, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_XBAR1_INOUT29** 0x400E81F0U, 0x4U, 0x400E86FCU, 0x1U, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_GPIO_MUX4_IO24** 0x400E81F0U, 0x5U, 0, 0, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_ENET_QOS_RX_DATA01** 0x400E81F0U, 0x8U, 0x400E84F4U, 0x0U, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_LPUART1_RXD** 0x400E81F0U, 0x9U, 0x400E861CU, 0x1U, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_03_GPIO10_IO24** 0x400E81F0U, 0xAU, 0, 0, 0x400E8434U
- **#define IOMUXC_GPIO_DISP_B1_04_VIDEO_MUX_LCDIF_DATA00** 0x400E81F4U, 0x0U, 0, 0, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_ENET_1G_RX_DATA02** 0x400E81F4U, 0x1U, 0x400E84D8U, 0x2U, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_LPUART4_RXD** 0x400E81F4U, 0x2U, 0, 0, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_TMR2_TIMER1** 0x400E81F4U, 0x3U, 0x400E864CU, 0x2U, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_XBAR1_INOUT30** 0x400E81F4U, 0x4U, 0x400E8700U, 0x1U, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_GPIO_MUX4_IO25** 0x400E81F4U, 0x5U, 0, 0, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_ENET_QOS_RX_DATA02** 0x400E81F4U, 0x8U, 0, 0, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_LPSPI3_SCK** 0x400E81F4U, 0x9U, 0x400E8600U, 0x1U, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_04_GPIO10_IO25** 0x400E81F4U, 0xAU, 0, 0, 0x400E8438U
- **#define IOMUXC_GPIO_DISP_B1_05_GPIO10_IO26** 0x400E81F8U, 0xAU, 0, 0, 0x400E843CU
- **#define IOMUXC_GPIO_DISP_B1_05_VIDEO_MUX_LCDIF_DATA01** 0x400E81F8U, 0x0U, 0, 0, 0x400E843CU
- **#define IOMUXC_GPIO_DISP_B1_05_ENET_1G_RX_DATA03** 0x400E81F8U, 0x1U, 0x400E84DCU, 0x2U, 0x400E843CU
- **#define IOMUXC_GPIO_DISP_B1_05_LPUART4_CTS_B** 0x400E81F8U, 0x2U, 0, 0, 0x400E843CU
- **#define IOMUXC_GPIO_DISP_B1_05_TMR2_TIMER2** 0x400E81F8U, 0x3U, 0x400E8650U,

- 0x1U, 0x400E843CU
- #define **IOMUXC_GPIO_DISP_B1_05_XBAR1_INOUT31** 0x400E81F8U, 0x4U, 0x400E8704U, 0x1U, 0x400E843CU
- #define **IOMUXC_GPIO_DISP_B1_05_GPIO_MUX4_IO26** 0x400E81F8U, 0x5U, 0, 0, 0x400E843CU
- #define **IOMUXC_GPIO_DISP_B1_05_ENET_QOS_RX_DATA03** 0x400E81F8U, 0x8U, 0, 0, 0x400E843CU
- #define **IOMUXC_GPIO_DISP_B1_05_LPSPI3_SIN** 0x400E81F8U, 0x9U, 0x400E8604U, 0x1U, 0x400E843CU
- #define **IOMUXC_GPIO_DISP_B1_06_VIDEO_MUX_LCDIF_DATA02** 0x400E81FCU, 0x0U, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_ENET_1G_TX_DATA03** 0x400E81FCU, 0x1U, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_LPUART4_TXD** 0x400E81FCU, 0x2U, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_TMR3_TIMER0** 0x400E81FCU, 0x3U, 0x400E8654U, 0x2U, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_XBAR1_INOUT32** 0x400E81FCU, 0x4U, 0x400E8708U, 0x1U, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_GPIO_MUX4_IO27** 0x400E81FCU, 0x5U, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_SRC_BT_CFG00** 0x400E81FCU, 0x6U, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_ENET_QOS_TX_DATA03** 0x400E81FCU, 0x8U, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_LPSPI3_SOUT** 0x400E81FCU, 0x9U, 0x400E8608U, 0x1U, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_06_GPIO10_IO27** 0x400E81FCU, 0xAU, 0, 0, 0x400E8440U
- #define **IOMUXC_GPIO_DISP_B1_07_VIDEO_MUX_LCDIF_DATA03** 0x400E8200U, 0x0U, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_ENET_1G_TX_DATA02** 0x400E8200U, 0x1U, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_LPUART4_RTS_B** 0x400E8200U, 0x2U, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_TMR3_TIMER1** 0x400E8200U, 0x3U, 0x400E8658U, 0x2U, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_XBAR1_INOUT33** 0x400E8200U, 0x4U, 0x400E870CU, 0x1U, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_GPIO_MUX4_IO28** 0x400E8200U, 0x5U, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_SRC_BT_CFG01** 0x400E8200U, 0x6U, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_ENET_QOS_TX_DATA02** 0x400E8200U, 0x8U, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_LPSPI3_PCS0** 0x400E8200U, 0x9U, 0x400E85F0U, 0x1U, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_07_GPIO10_IO28** 0x400E8200U, 0xAU, 0, 0, 0x400E8444U
- #define **IOMUXC_GPIO_DISP_B1_08_GPIO10_IO29** 0x400E8204U, 0xAU, 0, 0, 0x400E8444U

- E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_VIDEO_MUX_LCDIF_DATA04** 0x400E8204U, 0x0-U, 0, 0, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_ENET_1G_TX_DATA01** 0x400E8204U, 0x1U, 0, 0, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_USDHC1_CD_B** 0x400E8204U, 0x2U, 0x400E86C8U, 0x1U, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_TMR3_TIMER2** 0x400E8204U, 0x3U, 0x400E865CU, 0x1U, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_XBAR1_INOUT34** 0x400E8204U, 0x4U, 0x400E8710-U, 0x1U, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_GPIO_MUX4_IO29** 0x400E8204U, 0x5U, 0, 0, 0x400-E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_SRC_BT_CFG02** 0x400E8204U, 0x6U, 0, 0, 0x400-E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_ENET_QOS_TX_DATA01** 0x400E8204U, 0x8U, 0, 0, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_08_LPSPI3_PCS1** 0x400E8204U, 0x9U, 0x400E85F4U, 0x1U, 0x400E8448U
- #define **IOMUXC_GPIO_DISP_B1_09_VIDEO_MUX_LCDIF_DATA05** 0x400E8208U, 0x0-U, 0, 0, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_ENET_1G_TX_DATA00** 0x400E8208U, 0x1U, 0, 0, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_USDHC1_WP** 0x400E8208U, 0x2U, 0x400E86CCU, 0x1U, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_TMR4_TIMER0** 0x400E8208U, 0x3U, 0x400E8660U, 0x2U, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_XBAR1_INOUT35** 0x400E8208U, 0x4U, 0x400E8714-U, 0x1U, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_GPIO_MUX4_IO30** 0x400E8208U, 0x5U, 0, 0, 0x400-E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_SRC_BT_CFG03** 0x400E8208U, 0x6U, 0, 0, 0x400-E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_ENET_QOS_TX_DATA00** 0x400E8208U, 0x8U, 0, 0, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_LPSPI3_PCS2** 0x400E8208U, 0x9U, 0x400E85F8U, 0x1U, 0x400E844CU
- #define **IOMUXC_GPIO_DISP_B1_09_GPIO10_IO30** 0x400E8208U, 0xAU, 0, 0, 0x400E844-CU
- #define **IOMUXC_GPIO_DISP_B1_10_VIDEO_MUX_LCDIF_DATA06** 0x400E820CU, 0x0-U, 0, 0, 0x400E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_ENET_1G_TX_EN** 0x400E820CU, 0x1U, 0, 0, 0x400-E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_USDHC1_RESET_B** 0x400E820CU, 0x2U, 0, 0, 0x400E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_TMR4_TIMER1** 0x400E820CU, 0x3U, 0x400E8664U, 0x2U, 0x400E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_XBAR1_INOUT36** 0x400E820CU, 0x4U, 0, 0, 0x400-E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_GPIO_MUX4_IO31** 0x400E820CU, 0x5U, 0, 0, 0x400-

- E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_SRC_BT_CFG04** 0x400E820CU, 0x6U, 0, 0, 0x400-E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_ENET_QOS_TX_EN** 0x400E820CU, 0x8U, 0, 0, 0x400E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_LPSPI3_PCS3** 0x400E820CU, 0x9U, 0x400E85FCU, 0x1U, 0x400E8450U
- #define **IOMUXC_GPIO_DISP_B1_10_GPIO10_IO31** 0x400E820CU, 0xAU, 0, 0, 0x400-E8450U
- #define **IOMUXC_GPIO_DISP_B1_11_VIDEO_MUX_LCDIF_DATA07** 0x400E8210U, 0x0-U, 0, 0, 0x400E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_ENET_1G_TX_CLK_IO** 0x400E8210U, 0x1U, 0x400-E84E8U, 0x2U, 0x400E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_ENET_1G_REF_CLK** 0x400E8210U, 0x2U, 0x400-E84C4U, 0x2U, 0x400E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_TMR4_TIMER2** 0x400E8210U, 0x3U, 0x400E8668U, 0x1U, 0x400E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_XBAR1_INOUT37** 0x400E8210U, 0x4U, 0, 0, 0x400-E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_GPIO_MUX5_IO00** 0x400E8210U, 0x5U, 0, 0, 0x400-E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_SRC_BT_CFG05** 0x400E8210U, 0x6U, 0, 0, 0x400-E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_ENET_QOS_TX_CLK** 0x400E8210U, 0x8U, 0x400-E84A4U, 0x0U, 0x400E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_ENET_QOS_REF_CLK** 0x400E8210U, 0x9U, 0x400-E84A0U, 0x2U, 0x400E8454U
- #define **IOMUXC_GPIO_DISP_B1_11_GPIO11_IO00** 0x400E8210U, 0xAU, 0, 0, 0x400-E8454U
- #define **IOMUXC_GPIO_DISP_B2_00_GPIO11_IO01** 0x400E8214U, 0xAU, 0, 0, 0x400-E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_VIDEO_MUX_LCDIF_DATA08** 0x400E8214U, 0x0-U, 0, 0, 0x400E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_WDOG1_B** 0x400E8214U, 0x1U, 0, 0, 0x400E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_MQS_RIGHT** 0x400E8214U, 0x2U, 0, 0, 0x400E8458-U
- #define **IOMUXC_GPIO_DISP_B2_00_ENET_1G_TX_ER** 0x400E8214U, 0x3U, 0, 0, 0x400-E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_SAI1_TX_DATA03** 0x400E8214U, 0x4U, 0, 0, 0x400-E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_GPIO_MUX5_IO01** 0x400E8214U, 0x5U, 0, 0, 0x400-E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_SRC_BT_CFG06** 0x400E8214U, 0x6U, 0, 0, 0x400-E8458U
- #define **IOMUXC_GPIO_DISP_B2_00_ENET_QOS_TX_ER** 0x400E8214U, 0x8U, 0, 0, 0x400E8458U
- #define **IOMUXC_GPIO_DISP_B2_01_VIDEO_MUX_LCDIF_DATA09** 0x400E8218U, 0x0-U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_USDHC1_VSELECT** 0x400E8218U, 0x1U, 0, 0, 0x400E845CU

- #define **IOMUXC_GPIO_DISP_B2_01_MQS_LEFT** 0x400E8218U, 0x2U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_WDOG2_B** 0x400E8218U, 0x3U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_SAI1_TX_DATA02** 0x400E8218U, 0x4U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_GPIO_MUX5_IO02** 0x400E8218U, 0x5U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_SRC_BT_CFG07** 0x400E8218U, 0x6U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_EWM_OUT_B** 0x400E8218U, 0x8U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_CCM_ENET_REF_CLK_25M** 0x400E8218U, 0x9U, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_01_GPIO11_IO02** 0x400E8218U, 0xAU, 0, 0, 0x400E845CU
- #define **IOMUXC_GPIO_DISP_B2_02_GPIO11_IO03** 0x400E821CU, 0xAU, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_VIDEO_MUX_LCDIF_DATA10** 0x400E821CU, 0x0U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_ENET_TX_DATA00** 0x400E821CU, 0x1U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_PIT1_TRIGGER3** 0x400E821CU, 0x2U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_ARM_TRACE00** 0x400E821CU, 0x3U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_SAI1_TX_DATA01** 0x400E821CU, 0x4U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_GPIO_MUX5_IO03** 0x400E821CU, 0x5U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_SRC_BT_CFG08** 0x400E821CU, 0x6U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_02_ENET_QOS_TX_DATA00** 0x400E821CU, 0x8U, 0, 0, 0x400E8460U
- #define **IOMUXC_GPIO_DISP_B2_03_GPIO11_IO04** 0x400E8220U, 0xAU, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_VIDEO_MUX_LCDIF_DATA11** 0x400E8220U, 0x0U, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_ENET_TX_DATA01** 0x400E8220U, 0x1U, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_PIT1_TRIGGER2** 0x400E8220U, 0x2U, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_ARM_TRACE01** 0x400E8220U, 0x3U, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_SAI1_MCLK** 0x400E8220U, 0x4U, 0x400E866CU, 0x1U, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_GPIO_MUX5_IO04** 0x400E8220U, 0x5U, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_SRC_BT_CFG09** 0x400E8220U, 0x6U, 0, 0, 0x400E8464U
- #define **IOMUXC_GPIO_DISP_B2_03_ENET_QOS_TX_DATA01** 0x400E8220U, 0x8U, 0, 0, 0x400E8464U

- #define **IOMUXC_GPIO_DISP_B2_04_VIDEO_MUX_LCDIF_DATA12** 0x400E8224U, 0x0-U, 0, 0, 0x400E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_ENET_TX_EN** 0x400E8224U, 0x1U, 0, 0, 0x400-E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_PIT1_TRIGGER1** 0x400E8224U, 0x2U, 0, 0, 0x400-E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_ARM_TRACE02** 0x400E8224U, 0x3U, 0, 0, 0x400-E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_SAI1_RX_SYNC** 0x400E8224U, 0x4U, 0x400E8678U, 0x1U, 0x400E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_GPIO_MUX5_IO05** 0x400E8224U, 0x5U, 0, 0, 0x400-E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_SRC_BT_CFG10** 0x400E8224U, 0x6U, 0, 0, 0x400-E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_ENET_QOS_TX_EN** 0x400E8224U, 0x8U, 0, 0, 0x400E8468U
- #define **IOMUXC_GPIO_DISP_B2_04_GPIO11_IO05** 0x400E8224U, 0xAU, 0, 0, 0x400-E8468U
- #define **IOMUXC_GPIO_DISP_B2_05_GPIO11_IO06** 0x400E8228U, 0xAU, 0, 0, 0x400E846-CU
- #define **IOMUXC_GPIO_DISP_B2_05_VIDEO_MUX_LCDIF_DATA13** 0x400E8228U, 0x0-U, 0, 0, 0x400E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_ENET_TX_CLK** 0x400E8228U, 0x1U, 0x400E84C0U, 0x1U, 0x400E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_ENET_REF_CLK** 0x400E8228U, 0x2U, 0x400E84A8-U, 0x1U, 0x400E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_ARM_TRACE03** 0x400E8228U, 0x3U, 0, 0, 0x400-E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_SAI1_RX_BCLK** 0x400E8228U, 0x4U, 0x400E8670U, 0x1U, 0x400E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_GPIO_MUX5_IO06** 0x400E8228U, 0x5U, 0, 0, 0x400-E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_SRC_BT_CFG11** 0x400E8228U, 0x6U, 0, 0, 0x400-E846CU
- #define **IOMUXC_GPIO_DISP_B2_05_ENET_QOS_TX_CLK** 0x400E8228U, 0x8U, 0x400-E84A4U, 0x1U, 0x400E846CU
- #define **IOMUXC_GPIO_DISP_B2_06_GPIO11_IO07** 0x400E822CU, 0xAU, 0, 0, 0x400-E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_VIDEO_MUX_LCDIF_DATA14** 0x400E822CU, 0x0-U, 0, 0, 0x400E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_ENET_RX_DATA00** 0x400E822CU, 0x1U, 0x400E84-B0U, 0x1U, 0x400E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_LPUART7_TXD** 0x400E822CU, 0x2U, 0x400E8630U, 0x1U, 0x400E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_ARM_TRACE_CLK** 0x400E822CU, 0x3U, 0, 0, 0x400E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_SAI1_RX_DATA00** 0x400E822CU, 0x4U, 0x400-E8674U, 0x1U, 0x400E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_GPIO_MUX5_IO07** 0x400E822CU, 0x5U, 0, 0, 0x400-

- E8470U
- #define **IOMUXC_GPIO_DISP_B2_06_ENET_QOS_RX_DATA00** 0x400E822CU, 0x8U, 0x400E84F0U, 0x1U, 0x400E8470U
- #define **IOMUXC_GPIO_DISP_B2_07_VIDEO_MUX_LCDIF_DATA15** 0x400E8230U, 0x0-U, 0, 0, 0x400E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_ENET_RX_DATA01** 0x400E8230U, 0x1U, 0x400E84-B4U, 0x1U, 0x400E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_LPUART7_RXD** 0x400E8230U, 0x2U, 0x400E862CU, 0x1U, 0x400E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_ARM_TRACE_SWO** 0x400E8230U, 0x3U, 0, 0, 0x400E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_SAI1_TX_DATA00** 0x400E8230U, 0x4U, 0, 0, 0x400-E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_GPIO_MUX5_IO08** 0x400E8230U, 0x5U, 0, 0, 0x400-E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_ENET_QOS_RX_DATA01** 0x400E8230U, 0x8U, 0x400E84F4U, 0x1U, 0x400E8474U
- #define **IOMUXC_GPIO_DISP_B2_07_GPIO11_IO08** 0x400E8230U, 0xAU, 0, 0, 0x400-E8474U
- #define **IOMUXC_GPIO_DISP_B2_08_GPIO11_IO09** 0x400E8234U, 0xAU, 0, 0, 0x400-E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_VIDEO_MUX_LCDIF_DATA16** 0x400E8234U, 0x0-U, 0, 0, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_ENET_RX_EN** 0x400E8234U, 0x1U, 0x400E84B8U, 0x1U, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_LPUART8_TXD** 0x400E8234U, 0x2U, 0x400E8638U, 0x1U, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_ARM_CM7_EVENTO** 0x400E8234U, 0x3U, 0, 0, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_SAI1_TX_BCLK** 0x400E8234U, 0x4U, 0x400E867CU, 0x1U, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_GPIO_MUX5_IO09** 0x400E8234U, 0x5U, 0, 0, 0x400-E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_ENET_QOS_RX_EN** 0x400E8234U, 0x8U, 0x400E84-F8U, 0x1U, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_08_LPUART1_TXD** 0x400E8234U, 0x9U, 0x400E8620U, 0x2U, 0x400E8478U
- #define **IOMUXC_GPIO_DISP_B2_09_GPIO11_IO10** 0x400E8238U, 0xAU, 0, 0, 0x400E847-CU
- #define **IOMUXC_GPIO_DISP_B2_09_VIDEO_MUX_LCDIF_DATA17** 0x400E8238U, 0x0-U, 0, 0, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_ENET_RX_ER** 0x400E8238U, 0x1U, 0x400E84BCU, 0x1U, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_LPUART8_RXD** 0x400E8238U, 0x2U, 0x400E8634U, 0x1U, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_ARM_CM7_EVENTI** 0x400E8238U, 0x3U, 0, 0, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_SAI1_TX_SYNC** 0x400E8238U, 0x4U, 0x400E8680U, 0x1U, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_GPIO_MUX5_IO10** 0x400E8238U, 0x5U, 0, 0, 0x400-

- E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_ENET_QOS_RX_ER** 0x400E8238U, 0x8U, 0x400E84-FCU, 0x1U, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_09_LPUART1_RXD** 0x400E8238U, 0x9U, 0x400E861CU, 0x2U, 0x400E847CU
- #define **IOMUXC_GPIO_DISP_B2_10_GPIO11_IO11** 0x400E823CU, 0xAU, 0, 0, 0x400-E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_VIDEO_MUX_LCDIF_DATA18** 0x400E823CU, 0x0-U, 0, 0, 0x400E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_EMVSIM2_IO** 0x400E823CU, 0x1U, 0x400E86A8U, 0x1U, 0x400E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_LPUART2_TXD** 0x400E823CU, 0x2U, 0, 0, 0x400-E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_WDOG2_RESET_B_DEB** 0x400E823CU, 0x3U, 0, 0, 0x400E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_XBAR1_INOUT38** 0x400E823CU, 0x4U, 0, 0, 0x400-E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_GPIO_MUX5_IO11** 0x400E823CU, 0x5U, 0, 0, 0x400-E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_LPI2C3_SCL** 0x400E823CU, 0x6U, 0x400E85BCU, 0x1U, 0x400E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_ENET_QOS_RX_ER** 0x400E823CU, 0x8U, 0x400-E84FCU, 0x2U, 0x400E8480U
- #define **IOMUXC_GPIO_DISP_B2_10_SPDIF_IN** 0x400E823CU, 0x9U, 0x400E86B4U, 0x2U, 0x400E8480U
- #define **IOMUXC_GPIO_DISP_B2_11_VIDEO_MUX_LCDIF_DATA19** 0x400E8240U, 0x0-U, 0, 0, 0x400E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_EMVSIM2_CLK** 0x400E8240U, 0x1U, 0, 0, 0x400-E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_LPUART2_RXD** 0x400E8240U, 0x2U, 0, 0, 0x400-E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_WDOG1_RESET_B_DEB** 0x400E8240U, 0x3U, 0, 0, 0x400E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_XBAR1_INOUT39** 0x400E8240U, 0x4U, 0, 0, 0x400-E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_GPIO_MUX5_IO12** 0x400E8240U, 0x5U, 0, 0, 0x400-E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_LPI2C3_SDA** 0x400E8240U, 0x6U, 0x400E85C0U, 0x1U, 0x400E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_ENET_QOS_CRS** 0x400E8240U, 0x8U, 0, 0, 0x400-E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_SPDIF_OUT** 0x400E8240U, 0x9U, 0, 0, 0x400E8484U
- #define **IOMUXC_GPIO_DISP_B2_11_GPIO11_IO12** 0x400E8240U, 0xAU, 0, 0, 0x400-E8484U
- #define **IOMUXC_GPIO_DISP_B2_12_GPIO11_IO13** 0x400E8244U, 0xAU, 0, 0, 0x400-E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_VIDEO_MUX_LCDIF_DATA20** 0x400E8244U, 0x0-U, 0, 0, 0x400E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_EMVSIM2_RST** 0x400E8244U, 0x1U, 0, 0, 0x400-E8488U

- #define **IOMUXC_GPIO_DISP_B2_12_FLEXCAN1_TX** 0x400E8244U, 0x2U, 0, 0, 0x400-E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_LPUART2_CTS_B** 0x400E8244U, 0x3U, 0, 0, 0x400-E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_XBAR1_INOUT40** 0x400E8244U, 0x4U, 0, 0, 0x400-E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_GPIO_MUX5_IO13** 0x400E8244U, 0x5U, 0, 0, 0x400-E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_LPI2C4_SCL** 0x400E8244U, 0x6U, 0x400E85C4U, 0x1U, 0x400E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_ENET_QOS_COL** 0x400E8244U, 0x8U, 0, 0, 0x400-E8488U
- #define **IOMUXC_GPIO_DISP_B2_12_LPSPI4_SCK** 0x400E8244U, 0x9U, 0x400E8610U, 0x1U, 0x400E8488U
- #define **IOMUXC_GPIO_DISP_B2_13_GPIO11_IO14** 0x400E8248U, 0xAU, 0, 0, 0x400E848-CU
- #define **IOMUXC_GPIO_DISP_B2_13_VIDEO_MUX_LCDIF_DATA21** 0x400E8248U, 0x0-U, 0, 0, 0x400E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_EMVSIM2_SVEN** 0x400E8248U, 0x1U, 0, 0, 0x400-E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_FLEXCAN1_RX** 0x400E8248U, 0x2U, 0x400E8498U, 0x1U, 0x400E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_LPUART2_RTS_B** 0x400E8248U, 0x3U, 0, 0, 0x400-E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_ENET_REF_CLK** 0x400E8248U, 0x4U, 0x400E84A8-U, 0x2U, 0x400E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_GPIO_MUX5_IO14** 0x400E8248U, 0x5U, 0, 0, 0x400-E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_LPI2C4_SDA** 0x400E8248U, 0x6U, 0x400E85C8U, 0x1U, 0x400E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_ENET_QOS_1588_EVENT0_OUT** 0x400E8248U, 0x8U, 0, 0, 0x400E848CU
- #define **IOMUXC_GPIO_DISP_B2_13_LPSPI4_SIN** 0x400E8248U, 0x9U, 0x400E8614U, 0x1-U, 0x400E848CU
- #define **IOMUXC_GPIO_DISP_B2_14_GPIO_MUX5_IO15** 0x400E824CU, 0x5U, 0, 0, 0x400-E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_FLEXCAN1_TX** 0x400E824CU, 0x6U, 0, 0, 0x400-E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_ENET_QOS_1588_EVENT0_IN** 0x400E824CU, 0x8-U, 0, 0, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_LPSPI4_SOUT** 0x400E824CU, 0x9U, 0x400E8618U, 0x1U, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_GPIO11_IO15** 0x400E824CU, 0xAU, 0, 0, 0x400-E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_VIDEO_MUX_LCDIF_DATA22** 0x400E824CU, 0x0-U, 0, 0, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_EMVSIM2_PD** 0x400E824CU, 0x1U, 0x400E86ACU, 0x1U, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_WDOG2_B** 0x400E824CU, 0x2U, 0, 0, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_VIDEO_MUX_EXT_DCIC1** 0x400E824CU, 0x3U, 0,

- 0, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_14_ENET_1G_REF_CLK** 0x400E824CU, 0x4U, 0x400E84C4U, 0x3U, 0x400E8490U
- #define **IOMUXC_GPIO_DISP_B2_15_VIDEO_MUX_LCDIF_DATA23** 0x400E8250U, 0x0U, 0, 0, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_EMVSIM2_POWER_FAIL** 0x400E8250U, 0x1U, 0x400E86B0U, 0x1U, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_WDOG1_B** 0x400E8250U, 0x2U, 0, 0, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_VIDEO_MUX_EXT_DCIC2** 0x400E8250U, 0x3U, 0, 0, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_PIT1_TRIGGER0** 0x400E8250U, 0x4U, 0, 0, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_GPIO_MUX5_IO16** 0x400E8250U, 0x5U, 0, 0, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_FLEXCAN1_RX** 0x400E8250U, 0x6U, 0x400E8498U, 0x2U, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_ENET_QOS_1588_EVENT0_AUX_IN** 0x400E8250U, 0x8U, 0, 0, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_LPSPI4_PCS0** 0x400E8250U, 0x9U, 0x400E860CU, 0x1U, 0x400E8494U
- #define **IOMUXC_GPIO_DISP_B2_15_GPIO11_IO16** 0x400E8250U, 0xAU, 0, 0, 0x400E8494U

Configuration

- static void [IOMUXC_SetPinMux](#) (uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t inputOnfield)
Sets the IOMUXC pin mux mode.
- static void [IOMUXC_SetPinConfig](#) (uint32_t muxRegister, uint32_t muxMode, uint32_t inputRegister, uint32_t inputDaisy, uint32_t configRegister, uint32_t configValue)
Sets the IOMUXC pin configuration.
- static void [IOMUXC_SetSaiMClkClockSource](#) (IOMUXC_GPR_Type *base, iomuxc_gpr_saimclk_t mclk, uint8_t clkSrc)
Sets IOMUXC general configuration for SAI MCLK selection.
- static void [IOMUXC_MQSEnterSoftwareReset](#) (IOMUXC_GPR_Type *base, bool enable)
Enters or exit MQS software reset.
- static void [IOMUXC_MQSEnable](#) (IOMUXC_GPR_Type *base, bool enable)
Enables or disables MQS.
- static void [IOMUXC_MQSConfig](#) (IOMUXC_GPR_Type *base, iomuxc_mqs_pwm_oversample_rate_t rate, uint8_t divider)
Configure MQS PWM oversampling rate compared with mclk and divider ratio control for mclk from hmclk.

8.2 Macro Definition Documentation

8.2.1 #define FSL_IOMUXC_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

8.3 Function Documentation

8.3.1 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.

8.3.2 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.

**8.3.3 static void IOMUXC_SetSaiMClkClockSource (IOMUXC_GPR_Type * *base*,
iomuxc_gpr_saimclk_t *mclk*, uint8_t *clkSrc*) [inline], [static]**

Parameters

<i>base</i>	The IOMUXC GPR base address.
<i>mclk</i>	The SAI MCLK.
<i>clkSrc</i>	The clock source. Take refer to register setting details for the clock source in RM.

**8.3.4 static void IOMUXC_MQSEnterSoftwareReset (IOMUXC_GPR_Type * *base*,
bool *enable*) [inline], [static]**

Parameters

<i>base</i>	The IOMUXC GPR base address.
<i>enable</i>	Enter or exit MQS software reset.

**8.3.5 static void IOMUXC_MQSEnable (IOMUXC_GPR_Type * *base*, bool *enable*)
[inline], [static]**

Parameters

<i>base</i>	The IOMUXC GPR base address.
<i>enable</i>	Enable or disable the MQS.

**8.3.6 static void IOMUXC_MQSCfg (IOMUXC_GPR_Type * *base*,
iomuxc_mqs_pwm_oversample_rate_t *rate*, uint8_t *divider*) [inline],
[static]**

Parameters

<i>base</i>	The IOMUXC GPR base address.
<i>rate</i>	The MQS PWM oversampling rate, refer to "iomuxc_mqs_pwm_oversample_rate_t".
<i>divider</i>	The divider ratio control for mclk from hclk. mclk freq = 1 / (divider + 1) * hclk freq.



Chapter 9

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 10

PGMC Driver

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

The Power Gating and Memory Controller(PGMC) block is a power management component that controls power gating of power domains and memory low power mode. There are four sub modules in the PGMC block, they are:

- Basic Power Controller(BPC): To control power domains with simple isolation and power switch;
- CPU Power Controller(CPC): To control CPU platform with complex power domain and sequence requirement;
- PMIC Power Controller(PPC): To control PMIC standby mode outside of the chip;
- Memory Interface(MIF): Designed for dedicated technology of memory;

For the BPC sub module, the APIs in "Basic Power Controller Related Interfaces" function group are provided. For the CPC sub module, the APIs in "CPU Power Controller Related Interfaces" function group are provided. For the PPC sub module, the APIs in "PMIC Power Related Interfaces" function group are provided. For the MIF sub module, the APIs in "MIF Module Related APIs" are provided.

Chapter 11

PMU Driver

11.1 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_lpsr_ana_ldo_config_t](#)
LPSR ANA LDO config. [More...](#)
- struct [pmu_static_lpsr_dig_config_t](#)
LPSR DIG LDO Config in Static/Software Mode. [More...](#)
- struct [pmu_snvs_dig_config_t](#)
SNVS 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...](#)
- struct [pmu_gpc_body_bias_config_t](#)
The stucture of body bias config in GPC mode. [More...](#)

Enumerations

- enum `_pmu_setpoint_map` {
`kPMU_SetPoint0` = 1UL << 0UL,
`kPMU_SetPoint1` = 1UL << 1UL,
`kPMU_SetPoint2` = 1UL << 2UL,
`kPMU_SetPoint3` = 1UL << 3UL,
`kPMU_SetPoint4` = 1UL << 4UL,
`kPMU_SetPoint5` = 1UL << 5UL,
`kPMU_SetPoint6` = 1UL << 6UL,
`kPMU_SetPoint7` = 1UL << 7UL,
`kPMU_SetPoint8` = 1UL << 8UL,
`kPMU_SetPoint9` = 1UL << 9UL,
`kPMU_SetPoint10` = 1UL << 10UL,
`kPMU_SetPoint11` = 1UL << 11UL,
`kPMU_SetPoint12` = 1UL << 12UL,
`kPMU_SetPoint13` = 1UL << 13UL,
`kPMU_SetPoint14` = 1UL << 14UL,
`kPMU_SetPoint15` = 1UL << 15UL }
System setpoints enumeration.
- enum `pmu_ldo_name_t` {
`kPMU_PllLdo` = 0U,
`kPMU_LpsrAnaLdo` = 1U,
`kPMU_LpsrDigLdo` = 2U,
`kPMU_SnvsDigLdo` = 3U }
The name of LDOs.
- enum `pmu_body_bias_name_t` {
`kPMU_RBB_SOC` = 0x0U,
`kPMU_RBB_LPSR` = 0x1U }
The name of body bias.
- 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_lpsr_ana_ldo_charge_pump_current_t` {
`kPMU_LpsrAnaChargePump300nA` = 0U,
`kPMU_LpsrAnaChargePump400nA` = 1U,
`kPMU_LpsrAnaChargePump500nA` = 2U,
`kPMU_LpsrAnaChargePump600nA` = 3U }
The enumeration of LPSR ANA LDO's charge pump current.
- enum `pmu_lpsr_ana_ldo_output_range_t` {
`kPMU_LpsrAnaLdoOutputFrom1P77To1P83` = 0U,
`kPMU_LpsrAnaLdoOutputFrom1P72To1P77` = 1U,

`kPMU_LpsrAnaLdoOutputFrom1P82To1P88 = 2U }`

The enumeration of LPSR ANA LDO's output range.

- `enum pmu_lpsr_dig_voltage_step_time_t {`
`kPMU_LpsrDigVoltageStepInc15us = 0x0U,`
`kPMU_LpsrDigVoltageStepInc25us = 0x1U,`
`kPMU_LpsrDigVoltageStepInc50us = 0x2U,`
`kPMU_LpsrDigVoltageStepInc100us = 0x3U }`

The enumeration of voltage step time for LPSR DIG LDO.

- `enum pmu_lpsr_dig_target_output_voltage_t {`
`kPMU_LpsrDigTargetStableVoltage0P631V = 0x0U,`
`kPMU_LpsrDigTargetStableVoltage0P65V = 0x1U,`
`kPMU_LpsrDigTargetStableVoltage0P67V = 0x2U,`
`kPMU_LpsrDigTargetStableVoltage0P689V = 0x3U,`
`kPMU_LpsrDigTargetStableVoltage0P709V = 0x4U,`
`kPMU_LpsrDigTargetStableVoltage0P728V = 0x5U,`
`kPMU_LpsrDigTargetStableVoltage0P748V = 0x6U,`
`kPMU_LpsrDigTargetStableVoltage0P767V = 0x7U,`
`kPMU_LpsrDigTargetStableVoltage0P786V = 0x8U,`
`kPMU_LpsrDigTargetStableVoltage0P806V = 0x9U,`
`kPMU_LpsrDigTargetStableVoltage0P825V = 0xAU,`
`kPMU_LpsrDigTargetStableVoltage0P845V = 0xBU,`
`kPMU_LpsrDigTargetStableVoltage0P864V = 0xCU,`
`kPMU_LpsrDigTargetStableVoltage0P883V = 0xDU,`
`kPMU_LpsrDigTargetStableVoltage0P903V = 0xEU,`
`kPMU_LpsrDigTargetStableVoltage0P922V = 0xFU,`
`kPMU_LpsrDigTargetStableVoltage0P942V = 0x10U,`
`kPMU_LpsrDigTargetStableVoltage0P961V = 0x11U,`
`kPMU_LpsrDigTargetStableVoltage0P981V = 0x12U,`
`kPMU_LpsrDigTargetStableVoltage1P0V = 0x13U,`
`kPMU_LpsrDigTargetStableVoltage1P019V = 0x14U,`
`kPMU_LpsrDigTargetStableVoltage1P039V = 0x15U,`
`kPMU_LpsrDigTargetStableVoltage1P058V = 0x16U,`
`kPMU_LpsrDigTargetStableVoltage1P078V = 0x17U,`
`kPMU_LpsrDigTargetStableVoltage1P097V = 0x18U,`
`kPMU_LpsrDigTargetStableVoltage1P117V = 0x19U,`
`kPMU_LpsrDigTargetStableVoltage1P136V = 0x1AU,`
`kPMU_LpsrDigTargetStableVoltage1P155V = 0x1BU,`
`kPMU_LpsrDigTargetStableVoltage1P175V = 0x1CU,`
`kPMU_LpsrDigTargetStableVoltage1P194V = 0x1DU,`
`kPMU_LpsrDigTargetStableVoltage1P214V = 0x1EU,`
`kPMU_LpsrDigTargetStableVoltage1P233V = 0x1FU }`

The target output voltage of LPSR DIG LDO.

- `enum pmu_snvs_dig_charge_pump_current_t {`
`kPMU_SnvsDigChargePump12P5nA = 0U,`
`kPMU_SnvsDigChargePump6P25nA = 1U,`

`kPMU_SnvsDigChargePump18P75nA = 2U }`

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

- `enum pmu_snvs_dig_discharge_resistor_value_t {`
`kPMU_SnvsDigDischargeResistor15K = 0U,`
`kPMU_SnvsDigDischargeResistor30K = 1U,`
`kPMU_SnvsDigDischargeResistor9K = 2U }`

The enumeration of the SNVS 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_WellBiasPowerFromLpsrDigLdo = 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_IP8_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 IP8 adjustment.

Driver version

- #define `FSL_PMU_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`
PMU driver version.

LDOs Control APIs

- void `PMU_SetPiLLdoControlMode` (ANADIG_PMU_Type *base, `pmu_control_mode_t` mode)
Selects the control mode of the PLL LDO.
- void `PMU_SwitchPiLLdoToGPCMode` (ANADIG_PMU_Type *base)
Switches the PLL LDO from Static/Software Mode to GPC/Hardware Mode.

- 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_SetLpsrAnaLdoControlMode](#) (ANADIG_LDO_SNVs_Type *base, [pmu_control_mode_t](#) mode)
Selects the control mode of the LPSR ANA LDO.
- void [PMU_StaticEnableLpsrAnaLdoBypassMode](#) (ANADIG_LDO_SNVs_Type *base, bool enable)
Sets the Bypass mode of the LPSR ANA LDO.
- static bool [PMU_StaticCheckLpsrAnaLdoBypassMode](#) (ANADIG_LDO_SNVs_Type *base)
Checks whether the LPSR ANA LDO is in bypass mode.
- void [PMU_StaticGetLpsrAnaLdoDefaultConfig](#) ([pmu_static_lpsr_ana_ldo_config_t](#) *config)
Fill the LPSR ANA LDO configuration structure with default settings.
- void [PMU_StaticLpsrAnaLdoInit](#) (ANADIG_LDO_SNVs_Type *base, const [pmu_static_lpsr_ana_ldo_config_t](#) *config)
Initialize the LPSR ANA LDO in Static/Software Mode.
- void [PMU_StaticLpsrAnaLdoDeinit](#) (ANADIG_LDO_SNVs_Type *base)
Disable the output of LPSR ANA LDO.
- void [PMU_SetLpsrDigLdoControlMode](#) (ANADIG_LDO_SNVs_Type *base, [pmu_control_mode_t](#) mode)
Selects the control mode of the LPSR DIG LDO.
- void [PMU_StaticEnableLpsrDigLdoBypassMode](#) (ANADIG_LDO_SNVs_Type *base, bool enable)
Turn on/off Bypass mode of the LPSR DIG LDO in Static/Software mode.
- static bool [PMU_StaticCheckLpsrDigLdoBypassMode](#) (ANADIG_LDO_SNVs_Type *base)
Checks whether the LPSR DIG LDO is in bypass mode.
- void [PMU_StaticGetLpsrDigLdoDefaultConfig](#) ([pmu_static_lpsr_dig_config_t](#) *config)
Gets the default configuration of LPSR DIG LDO.
- void [PMU_StaticLpsrDigLdoInit](#) (ANADIG_LDO_SNVs_Type *base, const [pmu_static_lpsr_dig_config_t](#) *config)
Initialize the LPSR DIG LDO in static mode.
- void [PMU_StaticLpsrDigLdoDeinit](#) (ANADIG_LDO_SNVs_Type *base)
Disable the LPSR DIG LDO.
- void [PMU_GPCSetLpsrDigLdoTargetVoltage](#) (uint32_t setpointMap, [pmu_lpsr_dig_target_output_voltage_t](#) voltageValue)
Sets the voltage step of LPSR DIG LDO in certain setpoint during GPC mode.
- void [PMU_GetSnvsDigLdoDefaultConfig](#) ([pmu_snvs_dig_config_t](#) *config)
Gets the default config of the SNVS DIG LDO.
- void [PMU_SnvsDigLdoInit](#) (ANADIG_LDO_SNVs_DIG_Type *base, [pmu_ldo_operate_mode_t](#) mode)
Initialize the SNVS DIG LDO.
- static void [PMU_SnvsDigLdoDeinit](#) (ANADIG_LDO_SNVs_DIG_Type *base)
Disable SNVS DIG LDO.
- void [PMU_GPCEnableLdo](#) ([pmu_ldo_name_t](#) name, uint32_t setpointMap)
Controls the ON/OFF of the selected LDO in certain setpoints with GPC mode.
- void [PMU_GPCSetLdoOperateMode](#) ([pmu_ldo_name_t](#) name, uint32_t setpointMap, [pmu_ldo_operate_mode_t](#) mode)
Sets the operating mode of the selected LDO in certain setpoints with GPC mode.
- void [PMU_GPCEnableLdoTrackingMode](#) ([pmu_ldo_name_t](#) name, uint32_t setpointMap)

- Controls the ON/OFF of the selected LDOs' Tracking mode in certain setpoints with GPC mode.
- void [PMU_GPCEnableLdoBypassMode](#) ([pmu_ldo_name_t](#) name, [uint32_t](#) setpointMap)
Controls the ON/OFF of the selected LDOs' Bypass mode in certain setpoints with GPC mode.
- void [PMU_GPCEnableLdoStandbyMode](#) ([pmu_ldo_name_t](#) name, [uint32_t](#) setpointMap)
When STBY assert, enable/disable the selected LDO enter it's Low power mode.

Bandgap Control APIs

- void [PMU_SetBandgapControlMode](#) ([ANADIG_PMU_Type](#) *base, [pmu_control_mode_t](#) mode)
Selects the control mode of the Bandgap Reference.
- void [PMU_SwitchBandgapToGPCMode](#) ([ANADIG_PMU_Type](#) *base)
Switches the Bandgap from Static/Software Mode to GPC/Hardware Mode.
- 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.
- static void [PMU_GPCEnableBandgap](#) ([ANADIG_PMU_Type](#) *base, [uint32_t](#) setpointMap)
Controls the ON/OFF of the Bandgap in certain setpoints with GPC mode.
- static void [PMU_GPCEnableBandgapStandbyMode](#) ([ANADIG_PMU_Type](#) *base, [uint32_t](#) setpointMap)
Controls the ON/OFF of the Bandgap's Standby mode in certain setpoints with GPC mode.

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_SetBodyBiasControlMode](#) ([ANADIG_PMU_Type](#) *base, [pmu_body_bias_name_t](#) name, [pmu_control_mode_t](#) mode)
Selects the control mode of the Body Bias.
- void [PMU_EnableBodyBias](#) ([ANADIG_PMU_Type](#) *base, [pmu_body_bias_name_t](#) name, bool enable)
Enables/disables the selected body bias.
- void [PMU_GPCEnableBodyBias](#) ([pmu_body_bias_name_t](#) name, [uint32_t](#) setpointMap)
Controls the ON/OFF of the selected body bias in certain setpoints with GPC mode.
- void [PMU_GPCEnableBodyBiasStandbyMode](#) ([pmu_body_bias_name_t](#) name, [uint32_t](#) setpointMap)
Controls the ON/OFF of the selected Body Bias' Wbias power switch in certain setpoints with GPC mode.
- void [PMU_GPCGetBodyBiasDefaultConfig](#) ([pmu_gpc_body_bias_config_t](#) *config)
Gets the default config of body bias in GPC mode.
- void [PMU_GPCSetBodyBiasConfig](#) ([pmu_body_bias_name_t](#) name, const [pmu_gpc_body_bias_config_t](#) *config)
Sets the config of the selected Body Bias in GPC mode.

11.2 Data Structure Documentation

11.2.1 struct pmu_static_lpsr_ana_ldo_config_t

Data Fields

- [pmu_ldo_operate_mode_t mode](#)
The operate mode of LPSR 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_lpsr_ana_ldo_config_t::mode**
- (2) **bool pmu_static_lpsr_ana_ldo_config_t::enable2mALoad**
 - **true** Enables 2mA loading to prevent overshoot;
 - **false** Disables 2mA loading.
- (3) **bool pmu_static_lpsr_ana_ldo_config_t::enable4mALoad**
 - **true** Enables 4mA loading to prevent dramatic voltage drop;
 - **false** Disables 4mA load.
- (4) **bool pmu_static_lpsr_ana_ldo_config_t::enable20uALoad**
 - **true** Enables 20uA loading to prevent overshoot;
 - **false** Disables 20uA load.
- (5) **bool pmu_static_lpsr_ana_ldo_config_t::enableStandbyMode**
 - **true** Enables Standby mode, if the STBY assert, the LPSR ANA LDO enter LP mode
 - **false** Disables Standby mode.

11.2.2 struct pmu_static_lpsr_dig_config_t

Data Fields

- bool [enableStableDetect](#)
Enable/Disable Stable Detect.
- [pmu_lpsr_dig_voltage_step_time_t voltageStepTime](#)
Step time.
- [pmu_lpsr_dig_target_output_voltage_t targetVoltage](#)

The target output voltage.

Field Documentation

- (1) `bool pmu_static_lpsr_dig_config_t::enableStableDetect`
 - **true** Enables Stable Detect.
 - **false** Disables Stable Detect.
- (2) `pmu_lpsr_dig_voltage_step_time_t pmu_static_lpsr_dig_config_t::voltageStepTime`
- (3) `pmu_lpsr_dig_target_output_voltage_t pmu_static_lpsr_dig_config_t::targetVoltage`

11.2.3 struct pmu_snvs_dig_config_t

Data Fields

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

Field Documentation

- (1) `pmu_ldo_operate_mode_t pmu_snvs_dig_config_t::mode`
- (2) `pmu_snvs_dig_charge_pump_current_t pmu_snvs_dig_config_t::chargePumpCurrent`
- (3) `pmu_snvs_dig_discharge_resistor_value_t pmu_snvs_dig_config_t::dischargeResistorValue`
- (4) `uint8_t pmu_snvs_dig_config_t::trimValue`
- (5) `bool pmu_snvs_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_snvs_dig_config_t::enableLdoStable`

11.2.4 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

11.2.5 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 [enablePowerDownOption](#): 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::enablePowerDownOption

- **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

11.2.6 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_IP8_adjustment_t adjustment](#)
Well bias adjustment IP8, please refer to [pmu_well_bias_IP8_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`

11.2.7 `struct pmu_gpc_body_bias_config_t`

Data Fields

- `uint8_t` [PWELLRegulatorSize](#)
The size of the PWELL Regulator.
- `uint8_t` [NWELLRegulatorSize](#)
The size of the NWELL Regulator.
- `uint8_t` [oscillatorSize](#)
The size of the oscillator bits.
- `uint8_t` [regulatorStrength](#)
The strength of the selected regulator.

Field Documentation

- (1) `uint8_t` `pmu_gpc_body_bias_config_t::PWELLRegulatorSize`
- (2) `uint8_t` `pmu_gpc_body_bias_config_t::NWELLRegulatorSize`
- (3) `uint8_t` `pmu_gpc_body_bias_config_t::oscillatorSize`
- (4) `uint8_t` `pmu_gpc_body_bias_config_t::regulatorStrength`

11.3 Macro Definition Documentation

11.3.1 `#define FSL_PMU_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

Version 2.1.0.

11.4 Enumeration Type Documentation

11.4.1 `enum _pmu_setpoint_map`

Enumerator

- `kPMU_SetPoint0` Set point 0.
- `kPMU_SetPoint1` Set point 1.
- `kPMU_SetPoint2` Set point 2.
- `kPMU_SetPoint3` Set point 3.
- `kPMU_SetPoint4` Set point 4.
- `kPMU_SetPoint5` Set point 5.
- `kPMU_SetPoint6` Set point 6.
- `kPMU_SetPoint7` Set point 7.

kPMU_SetPoint8 Set point 8.
kPMU_SetPoint9 Set point 9.
kPMU_SetPoint10 Set point 10.
kPMU_SetPoint11 Set point 11.
kPMU_SetPoint12 Set point 12.
kPMU_SetPoint13 Set point 13.
kPMU_SetPoint14 Set point 14.
kPMU_SetPoint15 Set point 15.

11.4.2 enum pmu_ldo_name_t

Enumerator

kPMU_PllLdo The PLL LDO in SOC domain.
kPMU_LpsrAnaLdo The LPSR ANA LDO in LPSR domain.
kPMU_LpsrDigLdo The LPSR DIG LDO in LPSR domain.
kPMU_SnvsDigLdo The SNVS DIG LDO in SNVS domain.

11.4.3 enum pmu_body_bias_name_t

Enumerator

kPMU_RBB_SOC The RBB implemented in SOC.
kPMU_RBB_LPSR The RBB implemented in LPSRMIX.

11.4.4 enum pmu_control_mode_t

Enumerator

kPMU_StaticMode Static/Software Control mode.
kPMU_GPCMode GPC/Hardware Control mode.

11.4.5 enum pmu_ldo_operate_mode_t

Enumerator

kPMU_LowPowerMode LDOs operate in Low power mode.
kPMU_HighPowerMode LDOs operate in High power mode.

11.4.6 enum pmu_lpsr_ana_ldo_charge_pump_current_t

Enumerator

<i>kPMU_LpsrAnaChargePump300nA</i>	The current of the charge pump is selected as 300nA.
<i>kPMU_LpsrAnaChargePump400nA</i>	The current of the charge pump is selected as 400nA.
<i>kPMU_LpsrAnaChargePump500nA</i>	The current of the charge pump is selected as 500nA.
<i>kPMU_LpsrAnaChargePump600nA</i>	The current of the charge pump is selected as 600nA.

11.4.7 enum pmu_lpsr_ana_ldo_output_range_t

Enumerator

<i>kPMU_LpsrAnaLdoOutputFrom1P77To1P83</i>	The output voltage varies from 1.77V to 1.83V.
<i>kPMU_LpsrAnaLdoOutputFrom1P72To1P77</i>	The output voltage varies from 1.72V to 1.77V.
<i>kPMU_LpsrAnaLdoOutputFrom1P82To1P88</i>	The output voltage varies from 1.82V to 1.88V.

11.4.8 enum pmu_lpsr_dig_voltage_step_time_t

Enumerator

<i>kPMU_LpsrDigVoltageStepInc15us</i>	LPSR DIG LDO voltage step time selected as 15us.
<i>kPMU_LpsrDigVoltageStepInc25us</i>	LPSR DIG LDO voltage step time selected as 25us.
<i>kPMU_LpsrDigVoltageStepInc50us</i>	LPSR DIG LDO voltage step time selected as 50us.
<i>kPMU_LpsrDigVoltageStepInc100us</i>	LPSR DIG LDO voltage step time selected as 100us.

11.4.9 enum pmu_lpsr_dig_target_output_voltage_t

Enumerator

<i>kPMU_LpsrDigTargetStableVoltage0P631V</i>	The target voltage selected as 0.631V.
<i>kPMU_LpsrDigTargetStableVoltage0P65V</i>	The target voltage selected as 0.65V.
<i>kPMU_LpsrDigTargetStableVoltage0P67V</i>	The target voltage selected as 0.67V.
<i>kPMU_LpsrDigTargetStableVoltage0P689V</i>	The target voltage selected as 0.689V.
<i>kPMU_LpsrDigTargetStableVoltage0P709V</i>	The target voltage selected as 0.709V.
<i>kPMU_LpsrDigTargetStableVoltage0P728V</i>	The target voltage selected as 0.728V.
<i>kPMU_LpsrDigTargetStableVoltage0P748V</i>	The target voltage selected as 0.748V.
<i>kPMU_LpsrDigTargetStableVoltage0P767V</i>	The target voltage selected as 0.767V.
<i>kPMU_LpsrDigTargetStableVoltage0P786V</i>	The target voltage selected as 0.786V.
<i>kPMU_LpsrDigTargetStableVoltage0P806V</i>	The target voltage selected as 0.806V.
<i>kPMU_LpsrDigTargetStableVoltage0P825V</i>	The target voltage selected as 0.825V.
<i>kPMU_LpsrDigTargetStableVoltage0P845V</i>	The target voltage selected as 0.845V.

<i>kPMU_LpsrDigTargetStableVoltage0P864V</i>	The target voltage selected as 0.864V.
<i>kPMU_LpsrDigTargetStableVoltage0P883V</i>	The target voltage selected as 0.883V.
<i>kPMU_LpsrDigTargetStableVoltage0P903V</i>	The target voltage selected as 0.903V.
<i>kPMU_LpsrDigTargetStableVoltage0P922V</i>	The target voltage selected as 0.922V.
<i>kPMU_LpsrDigTargetStableVoltage0P942V</i>	The target voltage selected as 0.942V.
<i>kPMU_LpsrDigTargetStableVoltage0P961V</i>	The target voltage selected as 0.961V.
<i>kPMU_LpsrDigTargetStableVoltage0P981V</i>	The target voltage selected as 0.981V.
<i>kPMU_LpsrDigTargetStableVoltage1P0V</i>	The target voltage selected as 1.0V.
<i>kPMU_LpsrDigTargetStableVoltage1P019V</i>	The target voltage selected as 1.019V.
<i>kPMU_LpsrDigTargetStableVoltage1P039V</i>	The target voltage selected as 1.039V.
<i>kPMU_LpsrDigTargetStableVoltage1P058V</i>	The target voltage selected as 1.058V.
<i>kPMU_LpsrDigTargetStableVoltage1P078V</i>	The target voltage selected as 1.078V.
<i>kPMU_LpsrDigTargetStableVoltage1P097V</i>	The target voltage selected as 1.097V.
<i>kPMU_LpsrDigTargetStableVoltage1P117V</i>	The target voltage selected as 1.117V.
<i>kPMU_LpsrDigTargetStableVoltage1P136V</i>	The target voltage selected as 1.136V.
<i>kPMU_LpsrDigTargetStableVoltage1P155V</i>	The target voltage selected as 1.155V.
<i>kPMU_LpsrDigTargetStableVoltage1P175V</i>	The target voltage selected as 1.175V.
<i>kPMU_LpsrDigTargetStableVoltage1P194V</i>	The target voltage selected as 1.194V.
<i>kPMU_LpsrDigTargetStableVoltage1P214V</i>	The target voltage selected as 1.214V.
<i>kPMU_LpsrDigTargetStableVoltage1P233V</i>	The target voltage selected as 1.233V.

11.4.10 enum pmu_snvs_dig_charge_pump_current_t

Enumerator

<i>kPMU_SnvsDigChargePump12P5nA</i>	The current of SNVS DIG LDO's charge pump is selected as 12.5nA.
<i>kPMU_SnvsDigChargePump6P25nA</i>	The current of SNVS DIG LDO's charge pump is selected as 6.25nA.
<i>kPMU_SnvsDigChargePump18P75nA</i>	The current of SNVS DIG LDO's charge pump is selected as 18.75nA.

11.4.11 enum pmu_snvs_dig_discharge_resistor_value_t

Enumerator

<i>kPMU_SnvsDigDischargeResistor15K</i>	The Discharge Resistor is selected as 15K ohm.
<i>kPMU_SnvsDigDischargeResistor30K</i>	The Discharge Resistor is selected as 30K ohm.
<i>kPMU_SnvsDigDischargeResistor9K</i>	The Discharge Resistor is selected as 9K ohm.

11.4.12 enum _pmu_static_bandgap_power_down_option

Enumerator

kPMU_PowerDownBandgapFully Fully power down the bandgap module.

kPMU_PowerDownVoltageReferenceOutputOnly Power down only the reference output section of the bandgap.

kPMU_PowerDownBandgapVBGUPDetector Power down the VBGUP detector of the bandgap without affecting any additional functionality.

11.4.13 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.

11.4.14 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.

11.4.15 enum pmu_well_bias_power_source_t

Enumerator

kPMU_WellBiasPowerFromLpsrDigLdo LPSR Dig LDO supplies the power stage and NWELL sampler.

kPMU_WellBiasPowerFromDCDC DCDC supplies the power stage and NWELL sampler.

11.4.16 enum pmu_bias_area_size_t

Enumerator

kPMU_180uA_6mm2At125C Imax = 180uA; Areamax-RVT = 6.00mm² at 125C.
kPMU_150uA_5mm2At125C Imax = 150uA; Areamax-RVT = 5.00mm² at 125C.
kPMU_120uA_4mm2At125C Imax = 120uA; Areamax-RVT = 4.00mm² at 125C.
kPMU_90uA_3mm2At125C Imax = 90uA; Areamax-RVT = 3.00mm² at 125C.
kPMU_60uA_2mm2At125C Imax = 60uA; Areamax-RVT = 2.00mm² 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.

11.4.17 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.

11.4.18 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.

11.4.19 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.

11.4.20 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.

11.5 Function Documentation

11.5.1 void PMU_SetPIILdoControlMode (ANADIG_PMU_Type * *base*, pmu_control_mode_t *mode*)

Parameters

<i>base</i>	PMU peripheral base address.
<i>mode</i>	The control mode of the PLL LDO. Please refer to pmu_control_mode_t .

11.5.2 void PMU_SwitchPIILdoToGPCMode (ANADIG_PMU_Type * *base*)

Parameters

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

11.5.3 void PMU_StaticEnablePIILdo (ANADIG_PMU_Type * *base*)

Parameters

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

11.5.4 void PMU_SetLpsrAnaLdoControlMode (ANADIG_LDO_SNVS_Type * *base*, pmu_control_mode_t *mode*)

Parameters

<i>base</i>	PMU peripheral base address.
<i>mode</i>	The control mode of the LPSR ANA LDO. Please refer to pmu_control_mode_t .

11.5.5 void PMU_StaticEnableLpsrAnaLdoBypassMode (ANADIG_LDO_SNVS_Type * *base*, bool *enable*)

Parameters

<i>base</i>	ANADIG_LDO_SNVS peripheral base address.
<i>enable</i>	Enable/Disable bypass mode. <ul style="list-style-type: none"> • true Enable LPSR ANA Bypass mode. • false Disable LPSR ANA Bypass mode.

11.5.6 static bool PMU_StaticCheckLpsrAnaLdoBypassMode (ANADIG_LDO_SNVS_Type * *base*) [inline], [static]

Parameters

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

Returns

The result used to indicates whether the LPSR ANA LDO is in bypass mode.

- **true** The LPSR ANA LDO is in bypass mode.
- **false** The LPSR ANA LDO not in bypass mode.

11.5.7 void PMU_StaticGetLpsrAnaLdoDefaultConfig (pmu_static_lpsr_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_LpsrAnaLdoDriverStrength0;
    config->brownOutDetectorConfig = kPMU_LpsrAnaLdoBrownOutDetectorDisable;
    config->chargePumpCurrent     = kPMU_LpsrAnaChargePump300nA;
    config->outputRange          = kPMU_LpsrAnaLdoOutputFrom1P77To1P83
    ;
*
```

Parameters

<i>config</i>	Pointer to the structure pmu_static_lpsr_ana_ldo_config_t .
---------------	---

11.5.8 void PMU_StaticLpsrAnaLdoInit (ANADIG_LDO_SNVS_Type * *base*, const pmu_static_lpsr_ana_ldo_config_t * *config*)

Parameters

<i>base</i>	ANADIG_LDO_SNVS peripheral base address.
<i>config</i>	Pointer to the structure pmu_static_lpsr_ana_ldo_config_t .

11.5.9 void PMU_StaticLpsrAnaLdoDeinit (ANADIG_LDO_SNVS_Type * *base*)

Parameters

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

11.5.10 void PMU_SetLpsrDigLdoControlMode (ANADIG_LDO_SNVS_Type * *base*, pmu_control_mode_t *mode*)

Parameters

<i>base</i>	PMU peripheral base address.
<i>mode</i>	The control mode of the LPSR DIG LDO. Please refer to pmu_control_mode_t .

11.5.11 void PMU_StaticEnableLpsrDigLdoBypassMode (ANADIG_LDO_SNVS_Type * *base*, bool *enable*)

Parameters

<i>base</i>	ANADIG_LDO_SNVS peripheral base address.
<i>enable</i>	<ul style="list-style-type: none"> • true Turns on Bypass mode of the LPSR DIG LDO. • false Turns off Bypass mode of the LPSR DIG LDO.

11.5.12 static bool PMU_StaticCheckLpsrDigLdoBypassMode (ANADIG_LDO_SNVS_Type * *base*) [inline], [static]

Parameters

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

Returns

The result used to indicates whether the LPSR DIG LDO is in bypass mode.

- **true** The LPSR DIG LDO is in bypass mode.
- **false** The LPSR DIG LDO not in bypass mode.

11.5.13 void PMU_StaticGetLpsrDigLdoDefaultConfig (pmu_static_lpsr_dig_config_t * *config*)

The default values are:

```

*      config->enableStableDetect = false;
*      config->voltageStepTime    = kPMU_LpsrDigVoltageStepInc50us;
*      config->brownOutConfig     = kPMU_LpsrDigBrownOutDisable;
*      config->targetVoltage      = kPMU_LpsrDigTargetStableVoltage1P0V
*
*      ;
*      config->mode               = kPMU_HighPowerMode;
*
```


Parameters

<i>config</i>	Pointer to the structure pmu_static_lpsr_dig_config_t .
---------------	---

11.5.14 void PMU_StaticLpsrDigLdoInit (ANADIG_LDO_SNVS_Type * *base*, const pmu_static_lpsr_dig_config_t * *config*)

Parameters

<i>base</i>	ANADIG_LDO_SNVS peripheral base address.
<i>config</i>	Pointer to the structure pmu_static_lpsr_dig_config_t .

11.5.15 void PMU_StaticLpsrDigLdoDeinit (ANADIG_LDO_SNVS_Type * *base*)

Parameters

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

11.5.16 void PMU_GPCSetLpsrDigLdoTargetVoltage (uint32_t *setpointMap*, pmu_lpsr_dig_target_output_voltage_t *voltageValue*)

Note

The function provides the feature to set the voltage step to different setpoints.

Parameters

<i>setpointMap</i>	The map of setpoints should be the OR'ed Value of _pmu_setpoint_map .
<i>voltageValue</i>	The voltage step to be set. See enumeration pmu_lpsr_dig_target_output_voltage_t .

11.5.17 void PMU_GetSnvsDigLdoDefaultConfig (pmu_snvs_dig_config_t * *config*)

The default values are:

```
* config->mode                = kPMU_LowPowerMode;
* config->chargePumpCurrent    = kPMU_SnvsDigChargePump12P5nA;
* config->dischargeResistorValue = kPMU_SnvsDigDischargeResistor15K;
```

```

*      config->trimValue           = 0U;
*      config->enablePullDown      = true;
*      config->enableLdoStable     = false;
*

```

Parameters

<i>config</i>	Pointer to pmu_snvs_dig_config_t .
---------------	--

11.5.18 void PMU_SnvsDigLdoInit (ANADIG_LDO_SNVS_DIG_Type * *base*, pmu_ldo_operate_mode_t *mode*)

Parameters

<i>base</i>	LDO SNVS DIG peripheral base address.
<i>mode</i>	Used to control LDO power mode, please refer to pmu_ldo_operate_mode_t .

11.5.19 void PMU_GPCEnableLdo (pmu_ldo_name_t *name*, uint32_t *setpointMap*)

Parameters

<i>name</i>	The name of the selected ldo. Please see enumeration pmu_ldo_name_t for details.
<i>setpointMap</i>	The map of setpoints should be the OR'ed Value of _pmu_setpoint_map , 1b'1 means enable specific ldo in that setpoint. For example, the code PMU_GPCEnableLdo(k-PMU_PllLdo, 0x1U) means to enable PLL LDO in setpoint 0 and disable PLL LDO in other setpoint.

11.5.20 void PMU_GPCLdoSetOperateMode (pmu_ldo_name_t *name*, uint32_t *setpointMap*, pmu_ldo_operate_mode_t *mode*)

Parameters

<i>name</i>	The name of the selected ldo. Please see enumeration pmu_ldo_name_t for details.
-------------	--

<i>setpointMap</i>	The map of setpoints should be the OR'ed Value of _pmu_setpoint_map .
<i>mode</i>	The operating mode of the selected ldo. Please refer to enumeration pmu_ldo_operate_mode_t for details.

11.5.21 void PMU_GPCEnableLdoTrackingMode (pmu_ldo_name_t *name*, uint32_t *setpointMap*)

Parameters

<i>name</i>	The name of the selected ldo. Please see enumeration pmu_ldo_name_t for details.
<i>setpointMap</i>	The map of setpoints that the LDO tracking mode will be enabled in those setpoints, this value should be the OR'ed Value of _pmu_setpoint_map .

11.5.22 void PMU_GPCEnableLdoBypassMode (pmu_ldo_name_t *name*, uint32_t *setpointMap*)

Parameters

<i>name</i>	The name of the selected ldo. Please see enumeration pmu_ldo_name_t for details.
<i>setpointMap</i>	The map of setpoints that the LDO bypass mode will be enabled in those setpoints, this value should be the OR'ed Value of _pmu_setpoint_map .

11.5.23 void PMU_GPCEnableLdoStandbyMode (pmu_ldo_name_t *name*, uint32_t *setpointMap*)

Parameters

<i>name</i>	The name of the selected ldo. Please see enumeration pmu_ldo_name_t for details.
<i>setpointMap</i>	The map of setpoints that the LDO low power mode will be enabled in those setpoints if STBY assert, this value should be the OR'ed Value of _pmu_setpoint_map .

11.5.24 void PMU_SetBandgapControlMode (ANADIG_PMU_Type * *base*, pmu_control_mode_t *mode*)

Parameters

<i>base</i>	PMU peripheral base address.
<i>mode</i>	The control mode of the Bandgap Reference. Please refer to pmu_control_mode_t .

11.5.25 void PMU_SwitchBandgapToGPCMode (ANADIG_PMU_Type * *base*)

Parameters

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

11.5.26 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.

11.5.27 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.

11.5.28 void PMU_StaticBandgapInit (const pmu_static_bandgap_config_t * *config*)

Parameters

<i>config</i>	Pointer to the structure pmu_static_bandgap_config_t .
---------------	--

11.5.29 static void PMU_GPCEnableBandgap (ANADIG_PMU_Type * *base*, uint32_t *setpointMap*) [inline], [static]

For example, the code PMU_GPCEnableBandgap(PMU, kPMU_SetPoint0 | kPMU_SetPoint1); means enable bandgap in setpoint0 and setpoint1 and disable bandgap in other setpoints.

Parameters

<i>base</i>	PMU peripheral base address.
<i>setpointMap</i>	The map of setpoints that the bandgap will be enabled in those setpoints, this parameter should be the OR'ed Value of _pmu_setpoint_map .

11.5.30 static void PMU_GPCEnableBandgapStandbyMode (ANADIG_PMU_Type * *base*, uint32_t *setpointMap*) [inline], [static]

Parameters

<i>base</i>	PMU peripheral base address.
<i>setpointMap</i>	The map of setpoints that the bandgap standby mode will be enabled in those setpoints, this value should be the OR'ed Value of _pmu_setpoint_map .

11.5.31 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 pmu_well_bias_config_t structure.

11.5.32 void PMU_GetWellBiasDefaultConfig (pmu_well_bias_config_t * *config*)

Parameters

<i>config</i>	The pointer to the pmu_well_bias_config_t structure.
---------------	--

11.5.33 void PMU_SetBodyBiasControlMode (ANADIG_PMU_Type * *base*, pmu_body_bias_name_t *name*, pmu_control_mode_t *mode*)

Parameters

<i>base</i>	PMU peripheral base address.
<i>name</i>	The name of the body bias. Please refer to pmu_body_bias_name_t .
<i>mode</i>	The control mode of the Body Bias. Please refer to pmu_control_mode_t .

11.5.34 void PMU_EnableBodyBias (ANADIG_PMU_Type * *base*, *pmu_body_bias_name_t name*, bool *enable*)

Parameters

<i>base</i>	PMU peripheral base address.
<i>name</i>	The name of the body bias to be turned on/off, please refer to pmu_body_bias_name_t .
<i>enable</i>	Used to turn on/off the specific body bias. <ul style="list-style-type: none"> • true Enable the selected body bias. • false Disable the selected body bias.

11.5.35 void PMU_GPCEnableBodyBias (*pmu_body_bias_name_t name*, uint32_t *setpointMap*)

Parameters

<i>name</i>	The name of the selected body bias. Please see enumeration pmu_body_bias_name_t for details.
<i>setpointMap</i>	The map of setpoints that the specific body bias will be enabled in those setpoints, this value should be the OR'ed Value of <code>_pmu_setpoint_map</code> .

11.5.36 void PMU_GPCEnableBodyBiasStandbyMode (*pmu_body_bias_name_t name*, uint32_t *setpointMap*)

Parameters

<i>name</i>	The name of the selected body bias. Please see the enumeration pmu_body_bias_name_t for details.
<i>setpointMap</i>	The map of setpoints that the specific body bias's wbias power switch will be turn on in those setpoints, this value should be the OR'ed Value of _pmu_setpoint_map .

11.5.37 void PMU_GPCGetBodyBiasDefaultConfig (pmu_gpc_body_bias_config_t * *config*)

Parameters

<i>config</i>	Pointer to structure pmu_gpc_body_bias_config_t .
---------------	---

11.5.38 void PMU_GPCSetBodyBiasConfig (pmu_body_bias_name_t *name*, const pmu_gpc_body_bias_config_t * *config*)

Parameters

<i>name</i>	The name of the selected body bias. Please see enumeration pmu_body_bias_name_t for details.
<i>config</i>	Pointer to structure pmu_gpc_body_bias_config_t .

Chapter 12

SOC_SRC Driver

12.1 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.

Data Structures

- struct `src_setpoint_authentication_t`
The structure of setpoint authentication. [More...](#)
- struct `src_domain_mode_authentication_t`
The stucture of domain mode authentication. [More...](#)

Enumerations

- enum `src_core_name_t` {
 `kSRC_CM7Core` = 0x1U,
 `kSRC_CM4Core` = 0x2U }
System core.
- enum `src_boot_fuse_selection_t` {
 `kSRC_SerialDownloaderBootFlow` = 0U,
 `kSRC_NormalBootFlow` = 1U }
The enumeration of the boot fuse selection.
- enum `src_global_system_reset_source_t` {


```

kSRC_WdogReset = 0U,
kSRC_Wdog3Reset = 2U,
kSRC_Wdog4Reset = 4U,
kSRC_M4LockUpReset = 6U,
kSRC_M7LockUpReset = 8U,
kSRC_M4RequestReset = 10U,
kSRC_M7RequestReset = 12U,
kSRC_TempsenseReset = 14U,
kSRC_CSUReset = 16U,
kSRC_JageSoftwareReset = 18U,
kSRC_OverVoltageReset = 20U }

```

The enumeration of global system reset sources.

- `enum_src_global_system_reset_status_flags` {


```

kSRC_M7CoreIppResetFlag = 1UL << 0UL,
kSRC_M7CoreM7RequestResetFlag = 1UL << 1UL,
kSRC_M7CoreM7LockUpResetFlag = 1UL << 2UL,
kSRC_M7CoreCSUResetFlag = 1UL << 3UL,
kSRC_M7CoreIppUserResetFlag = 1UL << 4UL,
kSRC_M7CoreWdogResetFlag = 1UL << 5UL,
kSRC_M7CoreJtagResetFlag = 1UL << 6UL,
kSRC_M7CoreJtagSWResetFlag = 1UL << 7UL,
kSRC_M7CoreWdog3ResetFlag = 1UL << 8UL,
kSRC_M7CoreWdog4ResetFlag = 1UL << 9UL,
kSRC_M7CoreTempsenseResetFlag = 1UL << 10UL,
kSRC_M7CoreM4RequestResetFlag = 1UL << 11UL,
kSRC_M7CoreM4LockUpResetFlag = 1UL << 12UL,
kSRC_M7CoreOverVoltageResetFlag = 1UL << 13UL,
kSRC_M7CoreCdogResetFlag = 1UL << 14UL,
kSRC_M4CoreIppResetFlag = 1UL << 16UL,
kSRC_M4CoreM4RequestResetFlag = 1UL << 17UL,
kSRC_M4CoreM4LockUpResetFlag = 1UL << 18UL,
kSRC_M4CoreCSUResetFlag = 1UL << 19UL,
kSRC_M4CoreIppUserResetFlag = 1UL << 20UL,
kSRC_M4CoreWdogResetFlag = 1UL << 21UL,
kSRC_M4CoreJtagResetFlag = 1UL << 22UL,
kSRC_M4CoreJtagSWResetFlag = 1UL << 23UL,
kSRC_M4CoreWdog3ResetFlag = 1UL << 24UL,
kSRC_M4CoreWdog4ResetFlag = 1UL << 25UL,
kSRC_M4CoreTempsenseResetFlag = 1UL << 26UL,
kSRC_M4CoreM7RequestResetFlag = 1UL << 27UL,
kSRC_M4CoreM7LockUpResetFlag = 1UL << 28UL,
kSRC_M4CoreOverVoltageResetFlag = 1UL << 29UL,
kSRC_M4CoreCdogResetFlag = 1UL << 30UL }

```

The enumeration of reset status flags.

- `enum_src_global_system_reset_mode_t` {

```
kSRC_ResetSystem = 0x0U,
kSRC_DoNotResetSystem = 0x3U }
```

The enumeration of global system reset mode.

- enum `src_reset_slice_name_t` {


```
kSRC_MegaSlice = 0x0U,
kSRC_DisplaySlice = 0x1U,
kSRC_WakeUpSlice = 0x2U,
kSRC_LpsrSlice = 0x3U,
kSRC_M4CoreSlice = 0x4U,
kSRC_M7CoreSlice = 0x5U,
kSRC_M4DebugSlice = 0x6U,
kSRC_M7DebugSlice = 0x7U,
kSRC_Usbphy1Slice = 0x8U,
kSRC_Usbphy2Slice = 0x9U }
```

The enumeration of the slice name.

- enum `_src_domain_mode_selection` {


```
kSRC_Cpu0RunModeAssertReset = 1UL << 0UL,
kSRC_Cpu0WaitModeAssertReset = 1UL << 1UL,
kSRC_Cpu0StopModeAssertReset = 1UL << 2UL,
kSRC_Cpu0SuspendModeAssertReset = 1UL << 3UL,
kSRC_Cpu1RunModeAssertReset = 1UL << 4UL,
kSRC_Cpu1WaitModeAssertReset = 1UL << 5UL,
kSRC_Cpu1StopModeAssertReset = 1UL << 6UL,
kSRC_Cpu1SuspendModeAssertReset = 1UL << 7UL }
```

The enumeration of the domain mode.

- enum `_src_setpoint_selection` {


```
kSRC_SetPoint0AssertReset = 1UL << 0UL,
kSRC_SetPoint1AssertReset = 1UL << 1UL,
kSRC_SetPoint2AssertReset = 1UL << 2UL,
kSRC_SetPoint3AssertReset = 1UL << 3UL,
kSRC_SetPoint4AssertReset = 1UL << 4UL,
kSRC_SetPoint5AssertReset = 1UL << 5UL,
kSRC_SetPoint6AssertReset = 1UL << 6UL,
kSRC_SetPoint7AssertReset = 1UL << 7UL,
kSRC_SetPoint8AssertReset = 1UL << 8UL,
kSRC_SetPoint9AssertReset = 1UL << 9UL,
kSRC_SetPoint10AssertReset = 1UL << 10UL,
kSRC_SetPoint11AssertReset = 1UL << 11UL,
kSRC_SetPoint12AssertReset = 1UL << 12UL,
kSRC_SetPoint13AssertReset = 1UL << 13UL,
kSRC_SetPoint14AssertReset = 1UL << 14UL,
kSRC_SetPoint15AssertReset = 1UL << 15UL }
```

The enumeration of setpoint.

- enum `src_general_purpose_register_index_t` {

```

kSRC_GeneralPurposeRegister1 = 0U,
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,
kSRC_GeneralPurposeRegister20 }

```

The index of each general purpose register.

- enum `_src_slice_reset_source` {
`kSRC_SoftwareReset` = `SRC_SLICE_STAT_RST_BY_SW_MASK`,
`kSRC_PowerModeTransferReset` = `SRC_SLICE_STAT_RST_BY_HW_MASK` }

The enumeration of the reset source of each slice.

- enum `src_slice_reset_state_t` {
`kSRC_SliceResetFinished` = `0U`,
`kSRC_SliceResetInProgress` = `1U` }

The enumeration of the reset state of each slice.

Driver version

- #define `FSL_SRC_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 1)`)
SRC driver version 2.1.1.

Core Reset release

- void `SRC_ReleaseCoreReset` (`SRC_Type *base`, `src_core_name_t` coreName)
Releases related core reset operation.

Latched Boot Arguments Related Interfaces

- static uint32_t `SRC_GetBootConfig` (`SRC_Type *base`)
Gets Boot configuration.
- static uint8_t `SRC_GetBootMode` (`SRC_Type *base`)
Gets the latched state of the `BOOT_MODE1` and `BOOT_MODE0` signals.
- static `src_boot_fuse_selection_t` `SRC_GetBootFuseSelection` (`SRC_Type *base`)

- Gets the state of the BT_FUSE_SEL fuse.
- static uint8_t [SRC_GetSECConfigFuseState](#) (SRC_Type *base)
Gets the state of the SECCONFIG[1] fuse.

Global System Related Interfaces

- void [SRC_SetGlobalSystemResetMode](#) (SRC_Type *base, [src_global_system_reset_source_t](#) resetSource, [src_global_system_reset_mode_t](#) resetMode)
Sets the reset mode of global system reset source.
- static uint32_t [SRC_GetResetStatusFlags](#) (SRC_Type *base)
Gets global system reset status flags.
- static void [SRC_ClearGlobalSystemResetStatus](#) (SRC_Type *base, uint32_t mask)
Clears the status of global reset.

Slice Software Reset Related Interfaces

- void [SRC_AssertSliceSoftwareReset](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Asserts software reset for the selected slice.

Slice Low-Power Mode Transition Related Interfaces

- static void [SRC-AllowUserModeAccess](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, bool enable)
Allows/disallows user mode access.
- static void [SRC-AllowNonSecureModeAccess](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, bool enable)
Allows/disallows non secure mode access.
- static void [SRC_LockAccessSetting](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Locks the setting of user mode access and non secure mode access.
- static void [SRC_SetDomainIdWhiteList](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, uint8_t domainId)
Sets the domain ID white list for the selected slice.
- static void [SRC_LockDomainIdWhiteList](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Locks the value of white list.
- static void [SRC_SetAssignList](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, uint32_t assignList)
Sets the value of assign list.
- static void [SRC_LockAssignList](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Locks the value of assign list.
- static void [SRC_EnableSetPointTransferReset](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, bool enable)
Enable/disable setpoint transfer reset.
- static void [SRC_EnableDomainModeTransferReset](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, bool enable)
Enable/disable domain mode transfer reset.
- void [SRC_SetSliceSetPointConfig](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, uint32_t setpointConfig)
Sets setpoint configuration for the selected reset slice.
- void [SRC_SetSliceDomainModeConfig](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, uint32_t domainConfig)

Sets domain mode configuration for the selected reset slice.

- void [SRC_LockSliceMode](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Locks the value of SETPOINT_MODE and DOMAIN_MODE for the selected reset slice.

Get/Clear Slice Reset Status Flags

- static uint32_t [SRC_GetSliceResetStatusFlags](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Gets the reset status flags of the selected slice.
- static void [SRC_ClearSliceResetStatusFlags](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName, uint32_t mask)
Clears the reset status flags of the selected slice.

Get Slice Reset State

- [src_slice_reset_state_t](#) [SRC_GetSliceResetState](#) (SRC_Type *base, [src_reset_slice_name_t](#) sliceName)
Gets the reset state of the selected slice.

General Purpose Registers Related Interfaces

- static void [SRC_SetGeneralPurposeRegister](#) (SRC_Type *base, [src_general_purpose_register_index_t](#) index, uint32_t value)
Sets value to general purpose registers.
- static uint32_t [SRC_GetGeneralPurposeRegister](#) (SRC_Type *base, [src_general_purpose_register_index_t](#) index)
Gets the value from general purpose registers.

12.2 Data Structure Documentation

12.2.1 struct src_setpoint_authentication_t

Data Fields

- bool [enableSetpointTransferReset](#)
Control whether reset slice is in setpoint mode.
- uint32_t [whiteList](#)
Select the core to access set point control register.
- bool [lockWhiteList](#)
Control whether lock the value in white list.
- bool [lockSetting](#)
Control whether lock the setpoint access setting.
- bool [allowNonSecureModeAccess](#)
Allow both secure and non-secure modes to config setpoint.
- bool [allowUserModeAccess](#)
Allow both privilege and user modes to config setpoint.

Field Documentation**(1) bool src_setpoint_authentication_t::enableSetpointTransferReset**

- **true** Slice hardware reset will be triggered by set point transition.
- **false** Slice hardware reset will not be triggered by set point transition.

(2) uint32_t src_setpoint_authentication_t::whiteList

The logic OR'ed value of [src_core_name_t](#) enumeration.

(3) bool src_setpoint_authentication_t::lockSetting**(4) bool src_setpoint_authentication_t::allowNonSecureModeAccess****(5) bool src_setpoint_authentication_t::allowUserModeAccess****12.2.2 struct src_domain_mode_authentication_t****Data Fields**

- bool [enableDomainModeTransferReset](#)
Control whether reset slice is in domain mode.
- uint32_t [assignList](#)
Select the core that reset of slice would be subject to the selected core status transition.
- bool [lockAssignList](#)
Control whether lock the value in Assign list.

Field Documentation**(1) bool src_domain_mode_authentication_t::enableDomainModeTransferReset**

- **true** Slice hardware reset will be triggered by cpu power mode transition.
- **false** Slice hardware reset will not be triggered by cpu power mode transition.

(2) uint32_t src_domain_mode_authentication_t::assignList

The logic OR'ed value of [src_core_name_t](#) enumeration.

(3) bool src_domain_mode_authentication_t::lockAssignList**12.3 Macro Definition Documentation****12.3.1 #define FSL_SRC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))****12.4 Enumeration Type Documentation**

12.4.1 enum src_core_name_t

Enumerator

kSRC_CM7Core System Core CM4.

kSRC_CM4Core System Core CM7.

12.4.2 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.

12.4.3 enum src_global_system_reset_source_t

Enumerator

kSRC_WdogReset WDOG triggers the global system reset.

kSRC_Wdog3Reset WDOG3 triggers the global system reset.

kSRC_Wdog4Reset WDOG4 triggers the global system reset.

kSRC_M4LockUpReset M4 core lockup triggers the global system reset.

kSRC_M7LockUpReset M7 core lockup triggers the global system reset.

kSRC_M4RequestReset M4 core request triggers the global system reset.

kSRC_M7RequestReset M7 core request triggers the global system reset.

kSRC_TempsenseReset Tempsense triggers the global system reset.

kSRC_CSUReset CSU triggers the global system reset.

kSRC_JtagSoftwareReset JTAG software triggers the global system reset.

kSRC_OverVoltageReset Over voltage triggers the global system reset.

12.4.4 enum _src_global_system_reset_status_flags

Enumerator

kSRC_M7CoreIppResetFlag The M7 Core reset is the result of ipp_reset_b pin.

kSRC_M7CoreM7RequestResetFlag The M7 Core reset is the result of M7 core reset request.

kSRC_M7CoreM7LockUpResetFlag The M7 Core reset is the result of M7 core lock up.

kSRC_M7CoreCSUResetFlag The M7 Core reset is the result of csu_reset_b input.

kSRC_M7CoreIppUserResetFlag The M7 Core reset is the result of ipp_user_reset_b qualified reset.

kSRC_M7CoreWdogResetFlag The M7 Core reset is the result of the watchdog time-out event.

kSRC_M7CoreJtagResetFlag The M7 Core reset is the result of HIGH-Z reset from JTAG.

kSRC_M7CoreJtagSWResetFlag The M7 Core reset is the result of software reset from JTAG.

kSRC_M7CoreWdog3ResetFlag The M7 Core reset is the result of watchdog3 time-out event.

kSRC_M7CoreWdog4ResetFlag The M7 Core reset is the result of watchdog4 time-out event.

kSRC_M7CoreTempsenseResetFlag The M7 Core reset is the result of on-chip temperature sensor.

kSRC_M7CoreM4RequestResetFlag The M7 Core reset is the result of M4 CPU reset request.

kSRC_M7CoreM4LockUpResetFlag The M7 Core reset is the result of M4 CPU lock up.

kSRC_M7CoreOverVoltageResetFlag The M7 Core reset is the result of over voltage.

kSRC_M7CoreCdogResetFlag The M7 Core reset is the result of Cdog.

kSRC_M4CoreIppResetFlag The M4 Core reset is the result of ipp_reset_b pin.

kSRC_M4CoreM4RequestResetFlag The M4 Core reset is the result of M4 core reset request.

kSRC_M4CoreM4LockUpResetFlag The M4 Core reset is the result of M4 core lock up.

kSRC_M4CoreCSUResetFlag The M4 Core reset is the result of csu_reset_b input.

kSRC_M4CoreIppUserResetFlag The M4 Core reset is the result of ipp_user_reset_b qualified reset.

kSRC_M4CoreWdogResetFlag The M4 Core reset is the result of the watchdog time-out event.

kSRC_M4CoreJtagResetFlag The M4 Core reset is the result of HIGH-Z reset from JTAG.

kSRC_M4CoreJtagSWResetFlag The M4 Core reset is the result of software reset from JTAG.

kSRC_M4CoreWdog3ResetFlag The M4 Core reset is the result of watchdog3 time-out event.

kSRC_M4CoreWdog4ResetFlag The M4 Core reset is the result of watchdog4 time-out event.

kSRC_M4CoreTempsenseResetFlag The M4 Core reset is the result of on-chip temperature sensor.

kSRC_M4CoreM7RequestResetFlag The M4 Core reset is the result of M7 CPU reset request.

kSRC_M4CoreM7LockUpResetFlag The M4 Core reset is the result of M7 CPU lock up.

kSRC_M4CoreOverVoltageResetFlag The M4 Core reset is the result of over voltage.

kSRC_M4CoreCdogResetFlag The M4 Core reset is the result of Cdog.

12.4.5 enum src_global_system_reset_mode_t

Enumerator

kSRC_ResetSystem Generate the global system reset.

kSRC_DoNotResetSystem Do not generate the global system reset.

12.4.6 enum src_reset_slice_name_t

Enumerator

kSRC_MegaSlice Megamix reset slice.

kSRC_DisplaySlice Displaymix reset slice.

kSRC_WakeUpSlice Wakeupmix reset slice.

kSRC_LpsrSlice Lpsrmix reset slice.

kSRC_M4CoreSlice M4 core reset slice.

kSRC_M7CoreSlice M7 core reset slice.
kSRC_M4DebugSlice M4 debug reset slice.
kSRC_M7DebugSlice M7 debug reset slice.
kSRC_Usbphy1Slice USBPHY1 reset slice.
kSRC_Usbphy2Slice USBPHY2 reset slice.

12.4.7 enum _src_domain_mode_selection

Enumerator

kSRC_Cpu0RunModeAssertReset CPU0 in run mode will assert slice reset.
kSRC_Cpu0WaitModeAssertReset CPU0 in wait mode will assert reset.
kSRC_Cpu0StopModeAssertReset CPU0 in stop mode will assert reset.
kSRC_Cpu0SuspendModeAssertReset CPU0 in suspend mode will assert reset.
kSRC_Cpu1RunModeAssertReset CPU1 in run mode will assert slice reset.
kSRC_Cpu1WaitModeAssertReset CPU1 in wait mode will assert reset.
kSRC_Cpu1StopModeAssertReset CPU1 in stop mode will assert reset.
kSRC_Cpu1SuspendModeAssertReset CPU1 in suspend mode will assert reset.

12.4.8 enum _src_setpoint_selection

Enumerator

kSRC_SetPoint0AssertReset In setpoint0 will assert slice reset.
kSRC_SetPoint1AssertReset In setpoint1 will assert slice reset.
kSRC_SetPoint2AssertReset In setpoint2 will assert slice reset.
kSRC_SetPoint3AssertReset In setpoint3 will assert slice reset.
kSRC_SetPoint4AssertReset In setpoint4 will assert slice reset.
kSRC_SetPoint5AssertReset In setpoint5 will assert slice reset.
kSRC_SetPoint6AssertReset In setpoint6 will assert slice reset.
kSRC_SetPoint7AssertReset In setpoint7 will assert slice reset.
kSRC_SetPoint8AssertReset In setpoint8 will assert slice reset.
kSRC_SetPoint9AssertReset In setpoint9 will assert slice reset.
kSRC_SetPoint10AssertReset In setpoint10 will assert slice reset.
kSRC_SetPoint11AssertReset In setpoint11 will assert slice reset.
kSRC_SetPoint12AssertReset In setpoint12 will assert slice reset.
kSRC_SetPoint13AssertReset In setpoint13 will assert slice reset.
kSRC_SetPoint14AssertReset In setpoint14 will assert slice reset.
kSRC_SetPoint15AssertReset In setpoint15 will assert slice reset.

12.4.9 enum src_general_purpose_register_index_t

Enumerator

<i>kSRC_GeneralPurposeRegister1</i>	The index of General Purpose Register1.
<i>kSRC_GeneralPurposeRegister2</i>	The index of General Purpose Register2.
<i>kSRC_GeneralPurposeRegister3</i>	The index of General Purpose Register3.
<i>kSRC_GeneralPurposeRegister4</i>	The index of General Purpose Register4.
<i>kSRC_GeneralPurposeRegister5</i>	The index of General Purpose Register5.
<i>kSRC_GeneralPurposeRegister6</i>	The index of General Purpose Register6.
<i>kSRC_GeneralPurposeRegister7</i>	The index of General Purpose Register7.
<i>kSRC_GeneralPurposeRegister8</i>	The index of General Purpose Register8.
<i>kSRC_GeneralPurposeRegister9</i>	The index of General Purpose Register9.
<i>kSRC_GeneralPurposeRegister10</i>	The index of General Purpose Register10.
<i>kSRC_GeneralPurposeRegister11</i>	The index of General Purpose Register11.
<i>kSRC_GeneralPurposeRegister12</i>	The index of General Purpose Register12.
<i>kSRC_GeneralPurposeRegister13</i>	The index of General Purpose Register13.
<i>kSRC_GeneralPurposeRegister14</i>	The index of General Purpose Register14.
<i>kSRC_GeneralPurposeRegister15</i>	The index of General Purpose Register15.
<i>kSRC_GeneralPurposeRegister16</i>	The index of General Purpose Register16.
<i>kSRC_GeneralPurposeRegister17</i>	The index of General Purpose Register17.
<i>kSRC_GeneralPurposeRegister18</i>	The index of General Purpose Register18.
<i>kSRC_GeneralPurposeRegister19</i>	The index of General Purpose Register19.
<i>kSRC_GeneralPurposeRegister20</i>	The index of General Purpose Register20.

12.4.10 enum _src_slice_reset_source

Enumerator

<i>kSRC_SoftwareReset</i>	Reset is caused by software setting.
<i>kSRC_PowerModeTransferReset</i>	Reset is caused by the power mode transfer.

12.4.11 enum src_slice_reset_state_t

Enumerator

<i>kSRC_SliceResetFinished</i>	The reset is finished.
<i>kSRC_SliceResetInProgress</i>	The reset is in process.

12.5 Function Documentation

12.5.1 void SRC_ReleaseCoreReset (SRC_Type * *base*, src_core_name_t *coreName*)

The core reset will be held until the boot core to release it.

Parameters

<i>base</i>	SRC peripheral base address.
<i>coreName</i>	The name of the reset core to be released.

12.5.2 static uint32_t SRC_GetBootConfig (SRC_Type * *base*) [inline], [static]

Parameters

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

Returns

Boot configuration. Please refer to fusemap.

12.5.3 static uint8_t SRC_GetBootMode (SRC_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.

12.5.4 static src_boot_fuse_selection_t SRC_GetBootFuseSelection (SRC_Type * *base*) [inline], [static]

Parameters

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

Returns

The state of the BT_FUSE_SEL fuse, please refer to fusemap for more information.

12.5.5 `static uint8_t SRC_GetSECConfigFuseState (SRC_Type * base)`
`[inline], [static]`

Parameters

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

Returns

The state of the SECCONFIG[1] fuse. Please refer to fusemap for more information.

12.5.6 void SRC_SetGlobalSystemResetMode (SRC_Type * *base*, src_global_system_reset_source_t *resetSource*, src_global_system_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 src_global_system_reset_source_t for more details.
<i>resetMode</i>	The reset mode of each reset source. See src_global_system_reset_mode_t for more details.

12.5.7 static uint32_t SRC_GetResetStatusFlags (SRC_Type * *base*) [inline], [static]

Parameters

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

Returns

The status of global system reset status. See [_src_global_system_reset_status_flags](#) for more details.

12.5.8 static void SRC_ClearGlobalSystemResetStatus (SRC_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 _src_global_system_reset_status_flags for more details.

12.5.9 void SRC_AssertSliceSoftwareReset (SRC_Type * *base*, src_reset_slice_name_t *sliceName*)

Note

This function will return as soon as the reset is finished.

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice to be reset. See src_reset_slice_name_t for more details.

12.5.10 static void SRC-AllowUserModeAccess (SRC_Type * *base*, src_reset_slice_name_t *sliceName*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.
<i>enable</i>	Used to control user mode access. <ul style="list-style-type: none"> • true Allow user mode access. • false Disallow user mode access.

12.5.11 static void SRC-AllowNonSecureModeAccess (SRC_Type * *base*, src_reset_slice_name_t *sliceName*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.
<i>enable</i>	Used to control non secure mode access. <ul style="list-style-type: none"> • true Allow non secure mode access. • false Disallow non secure mode access.

12.5.12 static void SRC_LockAccessSetting (SRC_Type * *base*, src_reset_slice_name_t *sliceName*) [inline], [static]

Note

Once locked only reset can unlock related settings.

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.

12.5.13 static void SRC_SetDomainIdWhiteList (SRC_Type * *base*, src_reset_slice_name_t *sliceName*, uint8_t *domainId*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.
<i>domainId</i>	The core to access registers, should be the OR'ed value of src_core_name_t .

12.5.14 static void SRC_LockDomainIdWhiteList (SRC_Type * *base*, src_reset_slice_name_t *sliceName*) [inline], [static]

Note

Once locked only reset can unlock related settings.

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.

12.5.15 static void SRC_SetAssignList (SRC_Type * *base*, src_reset_slice_name_t *sliceName*, uint32_t *assignList*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.
<i>assignList</i>	Cores that subject to corresponding core status transition, should be the OR'ed value of src_core_name_t .

12.5.16 static void SRC_LockAssignList (SRC_Type * *base*, src_reset_slice_name_t *sliceName*) [inline], [static]

Note

Once locked only reset can unlock related settings.

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.

12.5.17 static void SRC_EnableSetPointTransferReset (SRC_Type * *base*, src_reset_slice_name_t *sliceName*, bool *enable*) [inline], [static]

Parameters

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

<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.
<i>enable</i>	User to control setpoint transfer reset. <ul style="list-style-type: none"> • true Enable setpoint transfer reset. • false Disable setpoint transfer reset.

**12.5.18 static void SRC_EnableDomainModeTransferReset (SRC_Type * *base*,
src_reset_slice_name_t *sliceName*, bool *enable*) [inline], [static]**

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice name to set, please refer to src_reset_slice_name_t for details.
<i>enable</i>	User to control domain mode reset. <ul style="list-style-type: none"> • true Enable domain mode reset. • false Disable domain mode reset.

**12.5.19 void SRC_SetSliceSetPointConfig (SRC_Type * *base*,
src_reset_slice_name_t *sliceName*, uint32_t *setpointConfig*)**

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The selected reset slice. See src_reset_slice_name_t for more details.
<i>setpointConfig</i>	The logic OR'ed value of _src_setpoint_selection enumeration, when the system in the selected setpoint slice reset will be asserted.

**12.5.20 void SRC_SetSliceDomainModeConfig (SRC_Type * *base*,
src_reset_slice_name_t *sliceName*, uint32_t *domainConfig*)**

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The selected reset slice. See src_reset_slice_name_t for more details.
<i>domainConfig</i>	The logic OR'ed value of _src_domain_mode_selection enumerations.

12.5.21 void SRC_LockSliceMode (SRC_Type * *base*, src_reset_slice_name_t *sliceName*)

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The selected reset slice. See src_reset_slice_name_t for more details.

12.5.22 static uint32_t SRC_GetSliceResetStatusFlags (SRC_Type * *base*, src_reset_slice_name_t *sliceName*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The slice to be reset. See src_reset_slice_name_t for more details.

Returns

The reset status flags for the selected slice. Please refer to [_src_slice_reset_source](#) for details.

12.5.23 static void SRC_ClearSliceResetStatusFlags (SRC_Type * *base*, src_reset_slice_name_t *sliceName*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The selected slice. See src_reset_slice_name_t for more details.
<i>mask</i>	The reset status flags to be cleared. Please refer to _src_slice_reset_source for more details.

12.5.24 src_slice_reset_state_t SRC_GetSliceResetState (SRC_Type * *base*, src_reset_slice_name_t *sliceName*)

Parameters

<i>base</i>	SRC peripheral base address.
<i>sliceName</i>	The selected slice. See src_reset_slice_name_t for more details.

Return values

<i>kSRC_SliceResetIn- Process</i>	The reset is in process.
<i>kSRC_SliceResetFinished</i>	The reset is finished.

**12.5.25 static void SRC_SetGeneralPurposeRegister (SRC_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 src_general_purpose_register_index_t .
<i>value</i>	Setting value for GPRx register.

**12.5.26 static uint32_t SRC_GetGeneralPurposeRegister (SRC_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 src_general_purpose_register_index_t .

Returns

The setting value for GPRx register.

Chapter 13

ACMP: Analog Comparator Driver

13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Comparator (ACMP) module of MCUXpresso SDK devices.

The ACMP driver is created to help the user operate the ACMP module better. This driver can be considered as a basic comparator with advanced features. The APIs for basic comparator can make the C-MP work as a general comparator, which compares the two input channel's voltage and creates the output of the comparator result immediately. The APIs for advanced feature can be used as the plug-in function based on the basic comparator, and can provide more ways to process the comparator's output.

13.2 Typical use case

13.2.1 Normal Configuration

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

13.2.2 Interrupt Configuration

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

13.2.3 Round robin Configuration

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

Data Structures

- struct [acmp_config_t](#)
Configuration for ACMP. [More...](#)
- struct [acmp_channel_config_t](#)
Configuration for channel. [More...](#)
- struct [acmp_filter_config_t](#)
Configuration for filter. [More...](#)
- struct [acmp_dac_config_t](#)
Configuration for DAC. [More...](#)
- struct [acmp_round_robin_config_t](#)
Configuration for round robin mode. [More...](#)
- struct [acmp_discrete_mode_config_t](#)
Configuration for discrete mode. [More...](#)

Macros

- #define `CMP_C0_CFx_MASK` (`CMP_C0_CFR_MASK` | `CMP_C0_CFF_MASK`)
The mask of status flags cleared by writing 1.

Enumerations

- enum `_acmp_interrupt_enable` {
`kACMP_OutputRisingInterruptEnable` = (1U << 0U),
`kACMP_OutputFallingInterruptEnable` = (1U << 1U),
`kACMP_RoundRobinInterruptEnable` = (1U << 2U) }
Interrupt enable/disable mask.
- enum `_acmp_status_flags` {
`kACMP_OutputRisingEventFlag` = `CMP_C0_CFR_MASK`,
`kACMP_OutputFallingEventFlag` = `CMP_C0_CFF_MASK`,
`kACMP_OutputAssertEventFlag` = `CMP_C0_COUT_MASK` }
Status flag mask.
- enum `acmp_hysteresis_mode_t` {
`kACMP_HysteresisLevel0` = 0U,
`kACMP_HysteresisLevel1` = 1U,
`kACMP_HysteresisLevel2` = 2U,
`kACMP_HysteresisLevel3` = 3U }
Comparator hard block hysteresis control.
- enum `acmp_reference_voltage_source_t` {
`kACMP_VrefSourceVin1` = 0U,
`kACMP_VrefSourceVin2` = 1U }
CMP Voltage Reference source.
- enum `acmp_fixed_port_t` {
`kACMP_FixedPlusPort` = 0U,
`kACMP_FixedMinusPort` = 1U }
Fixed mux port.
- enum `acmp_dac_work_mode_t` {
`kACMP_DACWorkLowSpeedMode` = 0U,
`kACMP_DACWorkHighSpeedMode` = 1U }
Internal DAC's work mode.
- enum `acmp_discrete_clock_source_t` {
`kACMP_DiscreteClockSlow` = 0U,
`kACMP_DiscreteClockFast` = 1U }
Discrete mode clock selection.
- enum `acmp_discrete_sample_time_t` {
`kACMP_DiscreteSampleTimeAs1T` = 0U,
`kACMP_DiscreteSampleTimeAs2T` = 1U,
`kACMP_DiscreteSampleTimeAs4T` = 2U,
`kACMP_DiscreteSampleTimeAs8T` = 3U,
`kACMP_DiscreteSampleTimeAs16T` = 4U,
`kACMP_DiscreteSampleTimeAs32T` = 5U,
`kACMP_DiscreteSampleTimeAs64T` = 6U,
`kACMP_DiscreteSampleTimeAs256T` = 7U }

- *ACMP discrete sample selection.*
- enum `acmp_discrete_phase_time_t` {
`kACMP_DiscretePhaseTimeAlt0` = 0U,
`kACMP_DiscretePhaseTimeAlt1` = 1U,
`kACMP_DiscretePhaseTimeAlt2` = 2U,
`kACMP_DiscretePhaseTimeAlt3` = 3U,
`kACMP_DiscretePhaseTimeAlt4` = 4U,
`kACMP_DiscretePhaseTimeAlt5` = 5U,
`kACMP_DiscretePhaseTimeAlt6` = 6U,
`kACMP_DiscretePhaseTimeAlt7` = 7U }
- *ACMP discrete phase time selection.*

Driver version

- #define `FSL_ACMP_DRIVER_VERSION` (`MAKE_VERSION(2U, 0U, 6U)`)
ACMP driver version 2.0.6.

Initialization and deinitialization

- void `ACMP_Init` (`CMP_Type *base`, const `acmp_config_t *config`)
Initializes the ACMP.
- void `ACMP_Deinit` (`CMP_Type *base`)
Deinitializes the ACMP.
- void `ACMP_GetDefaultConfig` (`acmp_config_t *config`)
Gets the default configuration for ACMP.

Basic Operations

- void `ACMP_Enable` (`CMP_Type *base`, bool enable)
Enables or disables the ACMP.
- void `ACMP_EnableLinkToDAC` (`CMP_Type *base`, bool enable)
Enables the link from CMP to DAC enable.
- void `ACMP_SetChannelConfig` (`CMP_Type *base`, const `acmp_channel_config_t *config`)
Sets the channel configuration.

Advanced Operations

- void `ACMP_EnableDMA` (`CMP_Type *base`, bool enable)
Enables or disables DMA.
- void `ACMP_EnableWindowMode` (`CMP_Type *base`, bool enable)
Enables or disables window mode.
- void `ACMP_SetFilterConfig` (`CMP_Type *base`, const `acmp_filter_config_t *config`)
Configures the filter.
- void `ACMP_SetDACConfig` (`CMP_Type *base`, const `acmp_dac_config_t *config`)
Configures the internal DAC.
- void `ACMP_SetRoundRobinConfig` (`CMP_Type *base`, const `acmp_round_robin_config_t *config`)
Configures the round robin mode.
- void `ACMP_SetRoundRobinPreState` (`CMP_Type *base`, `uint32_t` mask)
Defines the pre-set state of channels in round robin mode.

- static uint32_t [ACMP_GetRoundRobinStatusFlags](#) (CMP_Type *base)
Gets the channel input changed flags in round robin mode.
- void [ACMP_ClearRoundRobinStatusFlags](#) (CMP_Type *base, uint32_t mask)
Clears the channel input changed flags in round robin mode.
- static uint32_t [ACMP_GetRoundRobinResult](#) (CMP_Type *base)
Gets the round robin result.

Interrupts

- void [ACMP_EnableInterrupts](#) (CMP_Type *base, uint32_t mask)
Enables interrupts.
- void [ACMP_DisableInterrupts](#) (CMP_Type *base, uint32_t mask)
Disables interrupts.

Status

- uint32_t [ACMP_GetStatusFlags](#) (CMP_Type *base)
Gets status flags.
- void [ACMP_ClearStatusFlags](#) (CMP_Type *base, uint32_t mask)
Clears status flags.

Discrete mode

- void [ACMP_SetDiscreteModeConfig](#) (CMP_Type *base, const [acmp_discrete_mode_config_t](#) *config)
Configure the discrete mode.
- void [ACMP_GetDefaultDiscreteModeConfig](#) ([acmp_discrete_mode_config_t](#) *config)
Get the default configuration for discrete mode setting.

13.3 Data Structure Documentation

13.3.1 struct acmp_config_t

Data Fields

- [acmp_hysteresis_mode_t](#) hysteresisMode
Hysteresis mode.
- bool [enableHighSpeed](#)
Enable High Speed (HS) comparison mode.
- bool [enableInvertOutput](#)
Enable inverted comparator output.
- bool [useUnfilteredOutput](#)
Set compare output(COUT) to equal COUTA(true) or COUT(false).
- bool [enablePinOut](#)
The comparator output is available on the associated pin.

Field Documentation

(1) [acmp_hysteresis_mode_t](#) [acmp_config_t::hysteresisMode](#)

- (2) `bool acmp_config_t::enableHighSpeed`
- (3) `bool acmp_config_t::enableInvertOutput`
- (4) `bool acmp_config_t::useUnfilteredOutput`
- (5) `bool acmp_config_t::enablePinOut`

13.3.2 struct `acmp_channel_config_t`

The comparator's port can be input from channel mux or DAC. If port input is from channel mux, detailed channel number for the mux should be configured.

Data Fields

- `uint32_t plusMuxInput`
Plus mux input channel(0~7).
- `uint32_t minusMuxInput`
Minus mux input channel(0~7).

Field Documentation

- (1) `uint32_t acmp_channel_config_t::plusMuxInput`
- (2) `uint32_t acmp_channel_config_t::minusMuxInput`

13.3.3 struct `acmp_filter_config_t`

Data Fields

- `bool enableSample`
Using external SAMPLE as sampling clock input, or using divided bus clock.
- `uint32_t filterCount`
Filter Sample Count.
- `uint32_t filterPeriod`
Filter Sample Period.

Field Documentation

- (1) `bool acmp_filter_config_t::enableSample`
- (2) `uint32_t acmp_filter_config_t::filterCount`

Available range is 1-7, 0 would cause the filter disabled.

- (3) `uint32_t acmp_filter_config_t::filterPeriod`

The divider to bus clock. Available range is 0-255.

13.3.4 struct acmp_dac_config_t

Data Fields

- [acmp_reference_voltage_source_t](#) *referenceVoltageSource*
Supply voltage reference source.
- [uint32_t](#) *DACValue*
Value for DAC Output Voltage.

Field Documentation

(1) [acmp_reference_voltage_source_t](#) *acmp_dac_config_t::referenceVoltageSource*

(2) [uint32_t](#) *acmp_dac_config_t::DACValue*

Available range is 0-255.

13.3.5 struct acmp_round_robin_config_t

Data Fields

- [acmp_fixed_port_t](#) *fixedPort*
Fixed mux port.
- [uint32_t](#) *fixedChannelNumber*
Indicates which channel is fixed in the fixed mux port.
- [uint32_t](#) *checkerChannelMask*
Mask of checker channel index.
- [uint32_t](#) *sampleClockCount*
Specifies how many round-robin clock cycles(0~3) later the sample takes place.
- [uint32_t](#) *delayModulus*
Comparator and DAC initialization delay modulus.

Field Documentation

(1) [acmp_fixed_port_t](#) *acmp_round_robin_config_t::fixedPort*

(2) [uint32_t](#) *acmp_round_robin_config_t::fixedChannelNumber*

(3) [uint32_t](#) *acmp_round_robin_config_t::checkerChannelMask*

Available range is channel0:0x01 to channel7:0x80 for round-robin checker.

(4) [uint32_t](#) *acmp_round_robin_config_t::sampleClockCount*

(5) [uint32_t](#) *acmp_round_robin_config_t::delayModulus*

13.3.6 struct acmp_discrete_mode_config_t

Data Fields

- bool [enablePositiveChannelDiscreteMode](#)
Positive Channel Continuous Mode Enable.
- bool [enableNegativeChannelDiscreteMode](#)
Negative Channel Continuous Mode Enable.
- bool [enableResistorDivider](#)
Resistor Divider Enable is used to enable the resistor divider for the inputs when they come from 3v domain and their values are above 1.8v.
- [acmp_discrete_clock_source_t](#) [clockSource](#)
Select the clock source in order to generate the required timing for comparator to work in discrete mode.
- [acmp_discrete_sample_time_t](#) [sampleTime](#)
Select the ACMP total sampling time period.
- [acmp_discrete_phase_time_t](#) [phase1Time](#)
Select the ACMP phase 1 sampling time.
- [acmp_discrete_phase_time_t](#) [phase2Time](#)
Select the ACMP phase 2 sampling time.

Field Documentation

(1) bool acmp_discrete_mode_config_t::enablePositiveChannelDiscreteMode

By default, the continuous mode is used.

(2) bool acmp_discrete_mode_config_t::enableNegativeChannelDiscreteMode

By default, the continuous mode is used.

(3) bool acmp_discrete_mode_config_t::enableResistorDivider

(4) acmp_discrete_clock_source_t acmp_discrete_mode_config_t::clockSource

(5) acmp_discrete_sample_time_t acmp_discrete_mode_config_t::sampleTime

(6) acmp_discrete_phase_time_t acmp_discrete_mode_config_t::phase1Time

(7) acmp_discrete_phase_time_t acmp_discrete_mode_config_t::phase2Time

13.4 Macro Definition Documentation

13.4.1 #define FSL_ACMP_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 6U))

13.4.2 #define CMP_C0_CFx_MASK (CMP_C0_CFR_MASK | CMP_C0_CFF_MASK)

13.5 Enumeration Type Documentation

13.5.1 enum _acmp_interrupt_enable

Enumerator

- kACMP_OutputRisingInterruptEnable* Enable the interrupt when comparator outputs rising.
- kACMP_OutputFallingInterruptEnable* Enable the interrupt when comparator outputs falling.
- kACMP_RoundRobinInterruptEnable* Enable the Round-Robin interrupt.

13.5.2 enum _acmp_status_flags

Enumerator

- kACMP_OutputRisingEventFlag* Rising-edge on compare output has occurred.
- kACMP_OutputFallingEventFlag* Falling-edge on compare output has occurred.
- kACMP_OutputAssertEventFlag* Return the current value of the analog comparator output.

13.5.3 enum acmp_hysteresis_mode_t

See chip data sheet to get the actual hysteresis value with each level.

Enumerator

- kACMP_HysteresisLevel0* Offset is level 0 and Hysteresis is level 0.
- kACMP_HysteresisLevel1* Offset is level 0 and Hysteresis is level 1.
- kACMP_HysteresisLevel2* Offset is level 0 and Hysteresis is level 2.
- kACMP_HysteresisLevel3* Offset is level 0 and Hysteresis is level 3.

13.5.4 enum acmp_reference_voltage_source_t

Enumerator

- kACMP_VrefSourceVin1* Vin1 is selected as resistor ladder network supply reference Vin.
- kACMP_VrefSourceVin2* Vin2 is selected as resistor ladder network supply reference Vin.

13.5.5 enum acmp_fixed_port_t

Enumerator

- kACMP_FixedPlusPort* Only the inputs to the Minus port are swept in each round.
- kACMP_FixedMinusPort* Only the inputs to the Plus port are swept in each round.

13.5.6 enum acmp_dac_work_mode_t

Enumerator

kACMP_DACWorkLowSpeedMode DAC is selected to work in low speed and low power mode.

kACMP_DACWorkHighSpeedMode DAC is selected to work in high speed high power mode.

13.5.7 enum acmp_discrete_clock_source_t

Enumerator

kACMP_DiscreteClockSlow Slow clock (32kHz) is used as the discrete mode clock.

kACMP_DiscreteClockFast Fast clock (16-20MHz) is used as the discrete mode clock.

13.5.8 enum acmp_discrete_sample_time_t

These values configures the analog comparator sampling timing (specified by the discrete mode clock period T which is selected by [acmp_discrete_clock_source_t](#)) in discrete mode.

Enumerator

kACMP_DiscreteSampleTimeAs1T The sampling time equals to 1xT.

kACMP_DiscreteSampleTimeAs2T The sampling time equals to 2xT.

kACMP_DiscreteSampleTimeAs4T The sampling time equals to 4xT.

kACMP_DiscreteSampleTimeAs8T The sampling time equals to 8xT.

kACMP_DiscreteSampleTimeAs16T The sampling time equals to 16xT.

kACMP_DiscreteSampleTimeAs32T The sampling time equals to 32xT.

kACMP_DiscreteSampleTimeAs64T The sampling time equals to 64xT.

kACMP_DiscreteSampleTimeAs256T The sampling time equals to 256xT.

13.5.9 enum acmp_discrete_phase_time_t

There are two phases for sampling input signals, phase 1 and phase 2.

Enumerator

kACMP_DiscretePhaseTimeAlt0 The phase x active in one sampling selection 0.

kACMP_DiscretePhaseTimeAlt1 The phase x active in one sampling selection 1.

kACMP_DiscretePhaseTimeAlt2 The phase x active in one sampling selection 2.

kACMP_DiscretePhaseTimeAlt3 The phase x active in one sampling selection 3.

kACMP_DiscretePhaseTimeAlt4 The phase x active in one sampling selection 4.

kACMP_DiscretePhaseTimeAlt5 The phase x active in one sampling selection 5.

kACMP_DiscretePhaseTimeAlt6 The phase x active in one sampling selection 6.

kACMP_DiscretePhaseTimeAlt7 The phase x active in one sampling selection 7.

13.6 Function Documentation

13.6.1 void ACMP_Init (CMP_Type * *base*, const acmp_config_t * *config*)

The default configuration can be got by calling [ACMP_GetDefaultConfig\(\)](#).

Parameters

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

13.6.2 void ACMP_Deinit (CMP_Type * *base*)

Parameters

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

13.6.3 void ACMP_GetDefaultConfig (acmp_config_t * *config*)

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

Example:

```
config->enableHighSpeed = false;
config->enableInvertOutput = false;
config->useUnfilteredOutput = false;
config->enablePinOut = false;
config->enableHysteresisBothDirections = false;
config->hysteresisMode = kACMP_hysteresisMode0;
```

Parameters

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

13.6.4 void ACMP_Enable (CMP_Type * *base*, bool *enable*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>enable</i>	True to enable the ACMP.

13.6.5 void ACMP_EnableLinkToDAC (CMP_Type * *base*, bool *enable*)

When this bit is set, the DAC enable/disable is controlled by the bit CMP_C0[EN] instead of CMP_C1[D-ACEN].

Parameters

<i>base</i>	ACMP peripheral base address.
<i>enable</i>	Enable the feature or not.

13.6.6 void ACMP_SetChannelConfig (CMP_Type * *base*, const acmp_channel_config_t * *config*)

Note that the plus/minus mux's setting is only valid when the positive/negative port's input isn't from DAC but from channel mux.

Example:

```
acmp_channel_config_t configStruct = {0};
configStruct.positivePortInput = kACMP_PortInputFromDAC;
configStruct.negativePortInput = kACMP_PortInputFromMux;
configStruct.minusMuxInput = 1U;
ACMP_SetChannelConfig(CMP0, &configStruct);
```

Parameters

<i>base</i>	ACMP peripheral base address.
<i>config</i>	Pointer to channel configuration structure.

13.6.7 void ACMP_EnableDMA (CMP_Type * *base*, bool *enable*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>enable</i>	True to enable DMA.

13.6.8 void ACMP_EnableWindowMode (CMP_Type * *base*, bool *enable*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>enable</i>	True to enable window mode.

13.6.9 void ACMP_SetFilterConfig (CMP_Type * *base*, const acmp_filter_config_t * *config*)

The filter can be enabled when the filter count is bigger than 1, the filter period is greater than 0 and the sample clock is from divided bus clock or the filter is bigger than 1 and the sample clock is from external clock. Detailed usage can be got from the reference manual.

Example:

```
acmp_filter_config_t configStruct = {0};
configStruct.filterCount = 5U;
configStruct.filterPeriod = 200U;
configStruct.enableSample = false;
ACMP_SetFilterConfig(CMP0, &configStruct);
```

Parameters

<i>base</i>	ACMP peripheral base address.
<i>config</i>	Pointer to filter configuration structure.

13.6.10 void ACMP_SetDACConfig (CMP_Type * *base*, const acmp_dac_config_t * *config*)

Example:

```
acmp_dac_config_t configStruct = {0};
configStruct.referenceVoltageSource = kACMP_VrefSourceVin1;
configStruct.DACValue = 20U;
configStruct.enableOutput = false;
configStruct.workMode = kACMP_DACWorkLowSpeedMode;
ACMP_SetDACConfig(CMP0, &configStruct);
```

Parameters

<i>base</i>	ACMP peripheral base address.
<i>config</i>	Pointer to DAC configuration structure. "NULL" is for disabling the feature.

13.6.11 void ACMP_SetRoundRobinConfig (CMP_Type * *base*, const acmp_round_robin_config_t * *config*)

Example:

```
acmp_round_robin_config_t configStruct = {0};
configStruct.fixedPort = kACMP_FixedPlusPort;
configStruct.fixedChannelNumber = 3U;
configStruct.checkerChannelMask = 0xF7U;
configStruct.sampleClockCount = 0U;
configStruct.delayModulus = 0U;
ACMP_SetRoundRobinConfig(CMP0, &configStruct);
```

Parameters

<i>base</i>	ACMP peripheral base address.
<i>config</i>	Pointer to round robin mode configuration structure. "NULL" is for disabling the feature.

13.6.12 void ACMP_SetRoundRobinPreState (CMP_Type * *base*, uint32_t *mask*)

Note: The pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So get-round-robin-result can't return the same value as the value are set by pre-state.

Parameters

<i>base</i>	ACMP peripheral base address.
<i>mask</i>	Mask of round robin channel index. Available range is channel0:0x01 to channel7:0x80.

13.6.13 static uint32_t ACMP_GetRoundRobinStatusFlags (CMP_Type * *base*) [inline], [static]

Parameters

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

Returns

Mask of channel input changed asserted flags. Available range is channel0:0x01 to channel17:0x80.

13.6.14 void ACMP_ClearRoundRobinStatusFlags (CMP_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>mask</i>	Mask of channel index. Available range is channel0:0x01 to channel17:0x80.

13.6.15 static uint32_t ACMP_GetRoundRobinResult (CMP_Type * *base*) [inline], [static]

Note that the set-pre-state has different circuit with get-round-robin-result in the SOC even though they are same bits. So [ACMP_GetRoundRobinResult\(\)](#) can't return the same value as the value are set by [ACMP_SetRoundRobinPreState](#).

Parameters

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

Returns

Mask of round robin channel result. Available range is channel0:0x01 to channel17:0x80.

13.6.16 void ACMP_EnableInterrupts (CMP_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>mask</i>	Interrupts mask. See "_acmp_interrupt_enable".

13.6.17 void ACMP_DisableInterrupts (CMP_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>mask</i>	Interrupts mask. See "_acmp_interrupt_enable".

13.6.18 uint32_t ACMP_GetStatusFlags (CMP_Type * *base*)

Parameters

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

Returns

Status flags asserted mask. See "_acmp_status_flags".

13.6.19 void ACMP_ClearStatusFlags (CMP_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ACMP peripheral base address.
<i>mask</i>	Status flags mask. See "_acmp_status_flags".

13.6.20 void ACMP_SetDiscreteModeConfig (CMP_Type * *base*, const acmp_discrete_mode_config_t * *config*)

Configure the discrete mode when supporting 3V domain with 1.8V core.

Parameters

<i>base</i>	ACMP peripheral base address.
<i>config</i>	Pointer to configuration structure. See "acmp_discrete_mode_config_t".

13.6.21 void ACMP_GetDefaultDiscreteModeConfig (acmp_discrete_mode_config_t * *config*)

Parameters

<i>config</i>	Pointer to configuration structure to be restored with the setting values.
---------------	--

Chapter 14

ADC_ETC: ADC External Trigger Control

14.1 Overview

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

14.2 Typical use case

14.2.1 Software trigger Configuration

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

14.2.2 Hardware trigger Configuration

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

Data Structures

- struct [adc_etc_config_t](#)
ADC_ETC configuration. [More...](#)
- struct [adc_etc_trigger_chain_config_t](#)
ADC_ETC trigger chain configuration. [More...](#)
- struct [adc_etc_trigger_config_t](#)
ADC_ETC trigger configuration. [More...](#)

Macros

- #define [FSL_ADC_ETC_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 2, 1))
ADC_ETC driver version.
- #define [ADC_ETC_DMA_CTRL_TRGn_REQ_MASK](#) 0xFF0000U
The mask of status flags cleared by writing 1.

Enumerations

- enum [_adc_etc_status_flag_mask](#)
ADC_ETC customized status flags mask.
- enum [adc_etc_external_trigger_source_t](#)
External triggers sources.
- enum [adc_etc_interrupt_enable_t](#)
Interrupt enable/disable mask.

- enum [adc_etc_dma_mode_selection_t](#)
DMA mode selection.

Initialization

- void [ADC_ETC_Init](#) (ADC_ETC_Type *base, const [adc_etc_config_t](#) *config)
Initialize the ADC_ETC module.
- void [ADC_ETC_Deinit](#) (ADC_ETC_Type *base)
De-Initialize the ADC_ETC module.
- void [ADC_ETC_GetDefaultConfig](#) ([adc_etc_config_t](#) *config)
Gets an available pre-defined settings for the ADC_ETC's configuration.
- void [ADC_ETC_SetTriggerConfig](#) (ADC_ETC_Type *base, uint32_t triggerGroup, const [adc_etc_trigger_config_t](#) *config)
Set the external XBAR trigger configuration.
- void [ADC_ETC_SetTriggerChainConfig](#) (ADC_ETC_Type *base, uint32_t triggerGroup, uint32_t chainGroup, const [adc_etc_trigger_chain_config_t](#) *config)
Set the external XBAR trigger chain configuration.
- uint32_t [ADC_ETC_GetInterruptStatusFlags](#) (ADC_ETC_Type *base, [adc_etc_external_trigger_source_t](#) sourceIndex)
Gets the interrupt status flags of external XBAR and TSC triggers.
- void [ADC_ETC_ClearInterruptStatusFlags](#) (ADC_ETC_Type *base, [adc_etc_external_trigger_source_t](#) sourceIndex, uint32_t mask)
Clears the ADC_ETC's interrupt status falgs.
- static void [ADC_ETC_EnableDMA](#) (ADC_ETC_Type *base, uint32_t triggerGroup)
Enable the DMA corresponding to each trigger source.
- static void [ADC_ETC_DisableDMA](#) (ADC_ETC_Type *base, uint32_t triggerGroup)
Disable the DMA corresponding to each trigger sources.
- static uint32_t [ADC_ETC_GetDMAStatusFlags](#) (ADC_ETC_Type *base)
Get the DMA request status falgs.
- static void [ADC_ETC_ClearDMAStatusFlags](#) (ADC_ETC_Type *base, uint32_t mask)
Clear the DMA request status falgs.
- static void [ADC_ETC_DoSoftwareReset](#) (ADC_ETC_Type *base, bool enable)
When enable, all logical will be reset.
- static void [ADC_ETC_DoSoftwareTrigger](#) (ADC_ETC_Type *base, uint32_t triggerGroup)
Do software trigger corresponding to each XBAR trigger sources.
- uint32_t [ADC_ETC_GetADCCConversionValue](#) (ADC_ETC_Type *base, uint32_t triggerGroup, uint32_t chainGroup)
Get ADC conversion result from external XBAR sources.

14.3 Data Structure Documentation

14.3.1 struct [adc_etc_config_t](#)

14.3.2 struct [adc_etc_trigger_chain_config_t](#)

14.3.3 struct [adc_etc_trigger_config_t](#)

14.4 Macro Definition Documentation

14.4.1 #define FSL_ADC_ETC_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

Version 2.2.1.

14.4.2 #define ADC_ETC_DMA_CTRL_TRGn_REQ_MASK 0xFF0000U

14.5 Function Documentation

14.5.1 void ADC_ETC_Init (ADC_ETC_Type * *base*, const adc_etc_config_t * *config*)

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>config</i>	Pointer to "adc_etc_config_t" structure.

14.5.2 void ADC_ETC_Deinit (ADC_ETC_Type * *base*)

Parameters

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

14.5.3 void ADC_ETC_GetDefaultConfig (adc_etc_config_t * *config*)

This function initializes the ADC_ETC's configuration structure with available settings. The default values are:

```
* config->enableTSBypass = true;
* config->enableTSC0Trigger = false;
* config->enableTSC1Trigger = false;
* config->TSC0triggerPriority = 0U;
* config->TSC1triggerPriority = 0U;
* config->clockPreDivider = 0U;
* config->XBARtriggerMask = 0U;
*
```

Parameters

<i>config</i>	Pointer to "adc_etc_config_t" structure.
---------------	--

14.5.4 void ADC_ETC_SetTriggerConfig (ADC_ETC_Type * *base*, uint32_t *triggerGroup*, const adc_etc_trigger_config_t * *config*)

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>triggerGroup</i>	Trigger group index.
<i>config</i>	Pointer to "adc_etc_trigger_config_t" structure.

14.5.5 void ADC_ETC_SetTriggerChainConfig (ADC_ETC_Type * *base*, uint32_t *triggerGroup*, uint32_t *chainGroup*, const adc_etc_trigger_chain_config_t * *config*)

For example, if *triggerGroup* is set to 0U and *chainGroup* is set to 1U, which means Trigger0 source's chain1 would be configured.

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>triggerGroup</i>	Trigger group index. Available number is 0~7.
<i>chainGroup</i>	Trigger chain group index. Available number is 0~7.
<i>config</i>	Pointer to "adc_etc_trigger_chain_config_t" structure.

14.5.6 uint32_t ADC_ETC_GetInterruptStatusFlags (ADC_ETC_Type * *base*, adc_etc_external_trigger_source_t *sourceIndex*)

Parameters

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

<i>sourceIndex</i>	trigger source index.
--------------------	-----------------------

Returns

Status flags mask of trigger. Refer to "_adc_etc_status_flag_mask".

**14.5.7 void ADC_ETC_ClearInterruptStatusFlags (ADC_ETC_Type * *base*,
adc_etc_external_trigger_source_t *sourceIndex*, uint32_t *mask*)**

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>sourceIndex</i>	trigger source index.
<i>mask</i>	Status flags mask of trigger. Refer to "_adc_etc_status_flag_mask".

**14.5.8 static void ADC_ETC_EnableDMA (ADC_ETC_Type * *base*, uint32_t
triggerGroup) [inline], [static]**

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>triggerGroup</i>	Trigger group index. Available number is 0~7.

**14.5.9 static void ADC_ETC_DisableDMA (ADC_ETC_Type * *base*, uint32_t
triggerGroup) [inline], [static]**

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>triggerGroup</i>	Trigger group index. Available number is 0~7.

**14.5.10 static uint32_t ADC_ETC_GetDMAStatusFlags (ADC_ETC_Type * *base*)
[inline], [static]**

Only external XBAR sources support DMA request.

Parameters

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

Returns

Mask of external XBAR trigger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

**14.5.11 static void ADC_ETC_ClearDMAStatusFlags (ADC_ETC_Type * *base*,
uint32_t *mask*) [inline], [static]**

Only external XBAR sources support DMA request.

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>mask</i>	Mask of external XBAR trigger's DMA request asserted flags. Available range is trigger0:0x01 to trigger7:0x80.

**14.5.12 static void ADC_ETC_DoSoftwareReset (ADC_ETC_Type * *base*, bool
enable) [inline], [static]**

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>enable</i>	Enable/Disable the software reset.

**14.5.13 static void ADC_ETC_DoSoftwareTrigger (ADC_ETC_Type * *base*,
uint32_t *triggerGroup*) [inline], [static]**

Each XBAR trigger sources can be configured as HW or SW trigger mode. In hardware trigger mode, trigger source is from XBAR. In software mode, trigger source is from software trigger. TSC trigger sources can only work in hardware trigger mode.

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>triggerGroup</i>	Trigger group index. Available number is 0~7.

14.5.14 uint32_t ADC_ETC_GetADCCConversionValue (ADC_ETC_Type * *base*, uint32_t *triggerGroup*, uint32_t *chainGroup*)

For example, if *triggerGroup* is set to 0U and *chainGroup* is set to 1U, which means the API would return Trigger0 source's chain1 conversion result.

Parameters

<i>base</i>	ADC_ETC peripheral base address.
<i>triggerGroup</i>	Trigger group index. Available number is 0~7.
<i>chainGroup</i>	Trigger chain group index. Available number is 0~7.

Returns

ADC conversion result value.

Chapter 15

AIPSTZ: AHB to IP Bridge

15.1 Overview

The MCUXpresso SDK provides a driver for the AHB-to-IP Bridge (AIPSTZ) of MCUXpresso SDK devices.

Enumerations

- enum `aipstz_master_privilege_level_t` {
 `kAIPSTZ_MasterBufferedWriteEnable` = (1U << 3),
 `kAIPSTZ_MasterTrustedForReadEnable` = (1U << 2),
 `kAIPSTZ_MasterTrustedForWriteEnable` = (1U << 1),
 `kAIPSTZ_MasterForceUserModeEnable` = 1U }
 List of AIPSTZ privilege configuration.
- enum `aipstz_master_t`
 List of AIPSTZ masters.
- enum `aipstz_peripheral_access_control_t`
 List of AIPSTZ peripheral access control configuration.
- enum `aipstz_peripheral_t`
 List of AIPSTZ peripherals.

Driver version

- #define `FSL_AIPSTZ_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
 Version 2.0.1.

Initialization and deinitialization

- void `AIPSTZ_SetMasterPrivilegeLevel` (`AIPSTZ_Type *base`, `aipstz_master_t master`, `uint32_t privilegeConfig`)
 Configure the privilege level for master.
- void `AIPSTZ_SetPeripheralAccessControl` (`AIPSTZ_Type *base`, `aipstz_peripheral_t peripheral`, `uint32_t accessControl`)
 Configure the access for peripheral.

15.2 Enumeration Type Documentation

15.2.1 enum `aipstz_master_privilege_level_t`

Enumerator

`kAIPSTZ_MasterBufferedWriteEnable` Write accesses from this master are allowed to be buffered.

kAIPSTZ_MasterTrustedForReadEnable This master is trusted for read accesses.
kAIPSTZ_MasterTrustedForWriteEnable This master is trusted for write accesses.
kAIPSTZ_MasterForceUserModeEnable Accesses from this master are forced to user-mode.

15.2.2 enum aipstz_master_t

Organized by width for the 8-15 bits and shift for lower 8 bits.

15.2.3 enum aipstz_peripheral_access_control_t

15.2.4 enum aipstz_peripheral_t

Organized by register offset for higher 32 bits, width for the 8-15 bits and shift for lower 8 bits.

15.3 Function Documentation

15.3.1 void AIPSTZ_SetMasterPrivilegeLevel (AIPSTZ_Type * *base*, aipstz_master_t *master*, uint32_t *privilegeConfig*)

Parameters

<i>base</i>	AIPSTZ peripheral base pointer
<i>master</i>	Masters for AIPSTZ.
<i>privilegeConfig</i>	Configuration is ORed from aipstz_master_privilege_level_t .

15.3.2 void AIPSTZ_SetPeripheralAccessControl (AIPSTZ_Type * *base*, aipstz_peripheral_t *peripheral*, uint32_t *accessControl*)

Parameters

<i>base</i>	AIPSTZ peripheral base pointer
<i>peripheral</i>	Peripheral for AIPSTZ.

<i>accessControl</i>	Configuration is ORed from aipstz_peripheral_access_control_t .
----------------------	---

Chapter 16

AOI: Crossbar AND/OR/INVERT Driver

16.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar AND/OR/INVERT (AOI) block of MCUXpresso SDK devices.

The AOI module supports a configurable number of event outputs, where each event output represents a user-programmed combinational boolean function based on four event inputs. The key features of this module include:

- Four dedicated inputs for each event output
- User-programmable combinational boolean function evaluation for each event output
- Memory-mapped device connected to a slave peripheral (IPS) bus
- Configurable number of event outputs

16.2 Function groups

16.2.1 AOI Initialization

To initialize the AOI driver, call the [AOI_Init\(\)](#) function and pass a baseaddr pointer.

See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/aoi`.

16.2.2 AOI Get Set Operation

The AOI module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). The AOI is a highly programmable module for creating combinational boolean outputs for use as hardware triggers. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. To configure the selected AOI module event, call the API of the [AOI_SetEventLogicConfig\(\)](#) function. To get the current event state configure, call the API of [AOI_GetEventLogicConfig\(\)](#) function. The AOI module does not support any special modes of operation. See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/aoi`.

16.3 Typical use case

The AOI module is designed to be integrated in conjunction with one or more inter-peripheral crossbar switch (XBAR) modules. A crossbar switch is typically used to select the 4*n AOI inputs from among available peripheral outputs and GPIO signals. The n EVENTn outputs from the AOI module are typically

used as additional inputs to a second crossbar switch, adding to it the ability to connect to its outputs an arbitrary 4-input boolean function of its other inputs.

This is an example to initialize and configure the AOI driver for a possible use case. Because the AOI module function is directly connected with an XBAR (Inter-peripheral crossbar) module, other peripheral drivers (PIT, CMP, and XBAR) are used to show full functionality of AOI module.

For example: Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver-examples/aoi

Data Structures

- struct `aoi_event_config_t`
AOI event configuration structure. [More...](#)

Macros

- #define `AOI` `AOI0`
AOI peripheral address.

Enumerations

- enum `aoi_input_config_t` {
 `kAOI_LogicZero` = 0x0U,
 `kAOI_InputSignal` = 0x1U,
 `kAOI_InvInputSignal` = 0x2U,
 `kAOI_LogicOne` = 0x3U }
AOI input configurations.
- enum `aoi_event_t` {
 `kAOI_Event0` = 0x0U,
 `kAOI_Event1` = 0x1U,
 `kAOI_Event2` = 0x2U,
 `kAOI_Event3` = 0x3U }
AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

Driver version

- #define `FSL_AOI_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 1)`)
Version 2.0.1.

AOI Initialization

- void `AOI_Init` (`AOI_Type *base`)
Initializes an AOI instance for operation.
- void `AOI_Deinit` (`AOI_Type *base`)
Deinitializes an AOI instance for operation.

AOI Get Set Operation

- void [AOI_GetEventLogicConfig](#) (AOI_Type *base, [aoi_event_t](#) event, [aoi_event_config_t](#) *config)
Gets the Boolean evaluation associated.
- void [AOI_SetEventLogicConfig](#) (AOI_Type *base, [aoi_event_t](#) event, const [aoi_event_config_t](#) *eventConfig)
Configures an AOI event.

16.4 Data Structure Documentation

16.4.1 struct [aoi_event_config_t](#)

Defines structure [_aoi_event_config](#) and use the [AOI_SetEventLogicConfig\(\)](#) function to make whole event configuration.

Data Fields

- [aoi_input_config_t](#) PT0AC
Product term 0 input A.
- [aoi_input_config_t](#) PT0BC
Product term 0 input B.
- [aoi_input_config_t](#) PT0CC
Product term 0 input C.
- [aoi_input_config_t](#) PT0DC
Product term 0 input D.
- [aoi_input_config_t](#) PT1AC
Product term 1 input A.
- [aoi_input_config_t](#) PT1BC
Product term 1 input B.
- [aoi_input_config_t](#) PT1CC
Product term 1 input C.
- [aoi_input_config_t](#) PT1DC
Product term 1 input D.
- [aoi_input_config_t](#) PT2AC
Product term 2 input A.
- [aoi_input_config_t](#) PT2BC
Product term 2 input B.
- [aoi_input_config_t](#) PT2CC
Product term 2 input C.
- [aoi_input_config_t](#) PT2DC
Product term 2 input D.
- [aoi_input_config_t](#) PT3AC
Product term 3 input A.
- [aoi_input_config_t](#) PT3BC
Product term 3 input B.
- [aoi_input_config_t](#) PT3CC
Product term 3 input C.
- [aoi_input_config_t](#) PT3DC
Product term 3 input D.

16.5 Macro Definition Documentation

16.5.1 #define FSL_AOI_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

16.6 Enumeration Type Documentation

16.6.1 enum aoi_input_config_t

The selection item represents the Boolean evaluations.

Enumerator

kAOI_LogicZero Forces the input to logical zero.
kAOI_InputSignal Passes the input signal.
kAOI_InvInputSignal Inverts the input signal.
kAOI_LogicOne Forces the input to logical one.

16.6.2 enum aoi_event_t

Enumerator

kAOI_Event0 Event 0 index.
kAOI_Event1 Event 1 index.
kAOI_Event2 Event 2 index.
kAOI_Event3 Event 3 index.

16.7 Function Documentation

16.7.1 void AOI_Init (AOI_Type * *base*)

This function un-gates the AOI clock.

Parameters

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

16.7.2 void AOI_Deinit (AOI_Type * *base*)

This function shutdowns AOI module.

Parameters

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

16.7.3 void AOI_GetEventLogicConfig (AOI_Type * *base*, aoi_event_t *event*, aoi_event_config_t * *config*)

This function returns the Boolean evaluation associated.

Example:

```
aoi_event_config_t demoEventLogicStruct;

AOI_GetEventLogicConfig(AOI, kAOI_Event0, &demoEventLogicStruct);
```

Parameters

<i>base</i>	AOI peripheral address.
<i>event</i>	Index of the event which will be set of type aoi_event_t.
<i>config</i>	Selected input configuration .

16.7.4 void AOI_SetEventLogicConfig (AOI_Type * *base*, aoi_event_t *event*, const aoi_event_config_t * *eventConfig*)

This function configures an AOI event according to the aoiEventConfig structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event.

Example:

```
aoi_event_config_t demoEventLogicStruct;

demoEventLogicStruct.PT0AC = kAOI_InvInputSignal;
demoEventLogicStruct.PT0BC = kAOI_InputSignal;
demoEventLogicStruct.PT0CC = kAOI_LogicOne;
demoEventLogicStruct.PT0DC = kAOI_LogicOne;

demoEventLogicStruct.PT1AC = kAOI_LogicZero;
demoEventLogicStruct.PT1BC = kAOI_LogicOne;
demoEventLogicStruct.PT1CC = kAOI_LogicOne;
demoEventLogicStruct.PT1DC = kAOI_LogicOne;

demoEventLogicStruct.PT2AC = kAOI_LogicZero;
demoEventLogicStruct.PT2BC = kAOI_LogicOne;
demoEventLogicStruct.PT2CC = kAOI_LogicOne;
demoEventLogicStruct.PT2DC = kAOI_LogicOne;

demoEventLogicStruct.PT3AC = kAOI_LogicZero;
demoEventLogicStruct.PT3BC = kAOI_LogicOne;
demoEventLogicStruct.PT3CC = kAOI_LogicOne;
```

```
demoEventLogicStruct.PT3DC = kAOI_LogicOne;  
AOI_SetEventLogicConfig(AOI, kAOI_Event0, demoEventLogicStruct);
```

Parameters

<i>base</i>	AOI peripheral address.
<i>event</i>	Event which will be configured of type <code>aoi_event_t</code> .
<i>eventConfig</i>	Pointer to type <code>aoi_event_config_t</code> structure. The user is responsible for filling out the members of this structure and passing the pointer to this function.

Chapter 17

ASRC: Asynchronous sample rate converter

17.1 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](#)

17.2 ASRC Driver

17.2.1 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_ASRCCTR_AT_MASK](#)(index) ((uint32_t)1U << (ASRC_ASRCCTR_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_FPinWaitStateInterruptEnable = 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_StatusFPinWaitState = 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_StatusPairAOutput-
Ready,
kASRC_StatusPairBInterrupt = kASRC_StatusPairBInputReady | kASRC_StatusPairBOutput-
Ready,
kASRC_StatusPairCInterrupt = kASRC_StatusPairCInputReady | kASRC_StatusPairCOutput-
Ready }

```

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, 2)`)
Version 2.1.2.

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.

17.2.2 Data Structure Documentation

17.2.2.1 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

17.2.2.2 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](#)

17.2.2.3 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.

17.2.2.4 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.

17.2.2.5 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

17.2.3 Macro Definition Documentation

17.2.3.1 #define ASRC_XFER_QUEUE_SIZE (4U)

17.2.4 Enumeration Type Documentation

17.2.4.1 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_ASRCCLKConfigureFailed 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.

17.2.4.2 enum asrc_channel_pair_t

Enumerator

kASRC_ChannelPairA channel pair A value
kASRC_ChannelPairB channel pair B value
kASRC_ChannelPairC channel pair C value

17.2.4.3 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

17.2.4.4 anonymous enum

Enumerator

kASRC_FPInWaitStateInterruptEnable 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

17.2.4.5 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_StatusFPInWaitState 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

17.2.4.6 anonymous enum

Enumerator

kASRC_OutputFifoNearFull channel pair output fifo near full
kASRC_InputFifoNearEmpty channel pair input fifo near empty

17.2.4.7 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

17.2.4.8 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

17.2.4.9 enum asrc_data_width_t

Enumerator

kASRC_DataWidth24Bit data width 24bit
kASRC_DataWidth16Bit data width 16bit
kASRC_DataWidth8Bit data width 8bit

17.2.4.10 enum asrc_data_align_t

Enumerator

kASRC_DataAlignMSB data alignment MSB
kASRC_DataAlignLSB data alignment LSB

17.2.4.11 enum asrc_sign_extension_t

Enumerator

kASRC_NoSignExtension no sign extension
kASRC_SignExtension sign extension

17.2.5 Function Documentation

17.2.5.1 uint32_t ASRC_GetInstance (ASRC_Type * *base*)

Parameters

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

17.2.5.2 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.

17.2.5.3 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.
-------------	--------------------

17.2.5.4 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
-------------	-------------------

17.2.5.5 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.

17.2.5.6 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.
---------------	----------------------

17.2.5.7 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.
--------------	--------------------

17.2.5.8 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>outSample-Width</i>	output sample width.
<i>remainSamples</i>	output sample rate.

Return values

<i>remain</i>	samples number.
---------------	-----------------

17.2.5.9 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

17.2.5.10 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.

17.2.5.11 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 _asrc_interrupt_mask .

17.2.5.12 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 _asrc_interrupt_mask .

17.2.5.13 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.

17.2.5.14 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.

17.2.5.15 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.

**17.2.5.16 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.

**17.2.5.17 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.
--------------	-----------------

**17.2.5.18 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.

17.2.5.19 `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.

17.2.5.20 `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.

17.2.5.21 `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>inputSample-Rate</i>	input audio data sample rate.
<i>outputSample-Rate</i>	output audio data sample rate.

17.2.5.22 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

17.2.5.23 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.

17.2.5.24 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.
------------------------	--

17.2.5.25 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.

17.2.5.26 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.

17.2.5.27 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.

17.2.5.28 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 18

CAAM: Cryptographic Acceleration and Assurance Module

18.1 Overview

The MCUXpresso SDK provides the peripheral driver for the Cryptographic Acceleration and Assurance Module (CAAM) module. CAAM is a multi-functional accelerator that supports the cryptographic functions common in many security protocols. This includes AES128, AES256, DES, 3DES, SHA1, SHA224, SHA256, RSA-4096, and a random number generator with a true entropic seed. CAAM includes a DMA engine that is descriptor-based to reduce processor-accelerator interaction.

The driver comprises two sets of API functions.

In the first set, blocking APIs are provided for the selected subset of operations supported by CAAM hardware. The CAAM operations are complete, and results are made available for further usage, when a function returns. When called, these functions do not return until a CAAM operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status.

The CAAM job descriptor is placed on the system stack during the blocking API calls. The driver uses global variable to manage the input and output job rings. The driver uses critical section (implemented as global interrupt enable/disable) for a short time, whenever it needs to access these global variables. Therefore, the driver functions are designed to be re-entrant and as a consequence, one CPU thread can call one blocking API, such as AES Encrypt, while other CPU thread can call another blocking API, such as SHA-256 Update. The blocking functions provide typical interface to upper layer or application software.

In the second set, non-blocking variants of the first set APIs are provided. Internally, the blocking APIs are implemented as a non-blocking operation start, followed by a blocking wait (CPU polling CAAM output job ring). for an operation completion. The non-blocking functions allow upper layer to inject an application specific operation after the CAAM job start and CAAM job complete events. The RTOS event wait and RTOS event set can be an example of such an operation.

18.2 CAAM Driver Initialization and Configuration

The CAAM Job Ring interface is a software job programming interface. CAAM implements 2 Job Ring interfaces. The CAAM driver uses `caam_job_ring_interface_t` data type as the Job Ring interface. Job Ring interface 0 is mandatory to be configured for the CAAM driver, Job Ring interface 1 is optional.

Initialize CAAM after Power On Reset or reset cycle See the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam`.

The CAAM Driver is initialized by calling the `CAAM_Init()` function. It enables the CAAM module clock, it configures the Job Ring interface(s) and instantiates the CAAM RNG module in normal (non-deterministic) mode. Then, it calls `CAAM_RNG_GenerateSecureKey()` to load the JDKEK, TDKEK, and TDSK registers and finally configures the CAAM SCFGR register.

18.3 Comments about API usage in RTOS

CAAM operations provided by this driver are re-entrant by protecting global variables (Job Ring interface) in critical section (global interrupt enable/disable by `EnableGlobalIRQ()` and `DisableGlobalIRQ()` MCU-Xpresso SDK APIs). If required, different RTOS threads can call CAAM driver APIs simultaneously, given that `EnableGlobalIRQ()` and `DisableGlobalIRQ()` can create a critical section.

18.4 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time should be considered (interrupt latency of equal and lower priority interrupts increases).

18.5 Comments about DCACHE

Usually the output data are not cached, because CAAM driver cleans them before scheduling job and invalidate after that, but in some cases the computing can take longer time and data could be written to memory by CAAM after invalidate happens, so it's recommended to invalidate CAAM output data again, right before their usage.

18.6 CAAM Driver Examples

18.6.1 Simple examples

Encrypt plaintext by DES engine Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam` Encrypt plaintext by AES engine Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam` Compute keyed hash by AES engine (CMAC) Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam` Compute hash by MDHA engine (SHA-256) Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam` Compute modular integer exponentiation Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam` Compute elliptic curve point addition Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/caam`

Modules

- [CAAM Blocking APIs](#)
- [CAAM Non-blocking APIs](#)

Data Structures

- struct [caam_job_callback_t](#)
CAAM callback function. [More...](#)
- struct [caam_handle_t](#)
CAAM handle Specifies jobRing and optionally the user callback function. [More...](#)
- struct [caam_config_t](#)
CAAM configuration structure. [More...](#)

Typedefs

- typedef uint32_t [caam_desc_aes_ecb_t](#) [64]
Memory buffer to hold CAAM descriptor for AESA ECB job.
- typedef uint32_t [caam_desc_aes_cbc_t](#) [64]
Memory buffer to hold CAAM descriptor for AESA CBC job.
- typedef uint32_t [caam_desc_aes_ctr_t](#) [64]
Memory buffer to hold CAAM descriptor for AESA CTR job.
- typedef uint32_t [caam_desc_aes_ccm_t](#) [64]
Memory buffer to hold CAAM descriptor for AESA CCM job.
- typedef uint32_t [caam_desc_aes_gcm_t](#) [64]
Memory buffer to hold CAAM descriptor for AESA GCM job.
- typedef uint32_t [caam_desc_hash_t](#) [64]
Memory buffer to hold CAAM descriptor for MDHA job or AESA CMAC job.
- typedef uint32_t [caam_desc_rng_t](#) [64]
Memory buffer to hold CAAM descriptor for RNG jobs.
- typedef uint32_t [caam_desc_cipher_des_t](#) [64]
Memory buffer to hold CAAM descriptor for DESA jobs.
- typedef uint32_t [caam_desc_pkha_t](#) [64]
Memory buffer to hold CAAM descriptor for PKHA jobs.
- typedef uint32_t [caam_desc_pkha_ecc_t](#) [64]
Memory buffer to hold CAAM descriptor for PKHA ECC jobs.

Enumerations

- enum {
 [kStatus_CAAM_Again](#) = MAKE_STATUS(kStatusGroup_CAAM, 0),
 [kStatus_CAAM_DataOverflow](#) = MAKE_STATUS(kStatusGroup_CAAM, 1) }
CAAM status return codes.
- enum [caam_job_ring_t](#) {
 [kCAAM_JobRing0](#) = 0u,
 [kCAAM_JobRing1](#) = 1u,
 [kCAAM_JobRing2](#) = 2u,
 [kCAAM_JobRing3](#) = 3u }
CAAM job ring selection.
- enum [caam_wait_mode_t](#) {
 [kCAAM_Blocking](#) = 0u,
 [kCAAM_Nonblocking](#) = 1u }
CAAM driver wait mechanism.
- enum [caam_rng_sample_mode_t](#) {
 [kCAAM_RNG_SampleModeVonNeumann](#) = 0U,
 [kCAAM_RNG_SampleModeRaw](#) = 1U,
 [kCAAM_RNG_SampleModeVonNeumannRaw](#) }
CAAM RNG sample mode.
- enum [caam_rng_ring_osc_div_t](#) {
 [kCAAM_RNG_RingOscDiv0](#) = 0U,
 [kCAAM_RNG_RingOscDiv2](#) = 1U,
 [kCAAM_RNG_RingOscDiv4](#) = 2U,
 [kCAAM_RNG_RingOscDiv8](#) = 3U }
CAAM RNG ring oscillator divide.

- enum `caam_priblob_t` {
`kCAAM_PrivSecureBootBlobs` = 0U,
`kCAAM_PrivProvisioningBlobsType1` = 1U,
`kCAAM_PrivProvisioningBlobsType2` = 2U,
`kCAAM_NormalOperationBlobs` = 3U }
CAAM Private Blob.
- enum `caam_ext_key_xfr_source_t` {
`kCAAM_ExtKeyXfr_KeyRegisterClass1` = 1U,
`kCAAM_ExtKeyXfr_KeyRegisterClass2` = 2U,
`kCAAM_ExtKeyXfr_PkhaRamE` = 3U }
CAAM External Key Transfer command SRC (The source from which the key will be obtained)

Functions

- `status_t CAAM_Init` (CAAM_Type *base, const `caam_config_t` *config)
Initializes the CAAM driver.
- `status_t CAAM_Deinit` (CAAM_Type *base)
Deinitializes the CAAM driver.
- `void CAAM_GetDefaultConfig` (`caam_config_t` *config)
Gets the default configuration structure.
- `status_t CAAM_Wait` (CAAM_Type *base, `caam_handle_t` *handle, uint32_t *descriptor, `caam_wait_mode_t` mode)
Wait for a CAAM job to complete.
- `status_t CAAM_ExternalKeyTransfer` (CAAM_Type *base, `caam_handle_t` *handle, `caam_ext_key_xfr_source_t` keySource, size_t keySize)
External Key Transfer.

Driver version

- `#define FSL_CAAM_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 5))
CAAM driver version.

18.7 Data Structure Documentation

18.7.1 struct `caam_job_callback_t`

Data Fields

- `void(* JobCompleted)` (void *userData)
CAAM Job complete callback.

18.7.2 struct `caam_handle_t`

The user callback functions is invoked only if jobRing interrupt has been enabled by the user. By default the jobRing interrupt is disabled (default job complete test is polling CAAM output ring).

Data Fields

- [caam_job_callback_t](#) callback
Callback function.
- void * [userData](#)
Parameter for CAAM job complete callback.

18.7.3 struct caam_config_t

Data Fields

- [caam_rng_sample_mode_t](#) rngSampleMode
RTMCTL Sample Mode.
- [caam_rng_ring_osc_div_t](#) rngRingOscDiv
RTMCTL Oscillator Divide.
- bool [scfgrLockTrngProgramMode](#)
SCFGR Lock TRNG Program Mode.
- bool [scfgrEnableRandomDataBuffer](#)
SCFGR Enable random data buffer.
- bool [scfgrRandomRngStateHandle0](#)
SCFGR Random Number Generator State Handle 0.
- bool [scfgrRandomDpaResistance](#)
SCFGR Random Differential Power Analysis Resistance.
- [caam_priblob_t](#) [scfgrPriblob](#)
SCFGR Private Blob.

Field Documentation

- (1) [caam_rng_sample_mode_t](#) [caam_config_t::rngSampleMode](#)
- (2) [caam_rng_ring_osc_div_t](#) [caam_config_t::rngRingOscDiv](#)
- (3) bool [caam_config_t::scfgrLockTrngProgramMode](#)
- (4) bool [caam_config_t::scfgrEnableRandomDataBuffer](#)
- (5) bool [caam_config_t::scfgrRandomRngStateHandle0](#)
- (6) bool [caam_config_t::scfgrRandomDpaResistance](#)
- (7) [caam_priblob_t](#) [caam_config_t::scfgrPriblob](#)

18.8 Macro Definition Documentation

18.8.1 #define FSL_CAAM_DRIVER_VERSION (MAKE_VERSION(2, 1, 5))

Version 2.1.5.

Current version: 2.1.4

Change log:

- Version 2.0.0
 - Initial version
- Version 2.0.1
 - Add Job Ring 2 and 3.
- Version 2.0.2
 - Add Data and Instruction Synchronization Barrier in `caam_input_ring_set_jobs_added()` to make sure that the descriptor will be loaded into CAAM correctly.
- Version 2.0.3
 - Use MACRO instead of numbers in descriptor.
 - Correct descriptor size mask.
- Version 2.1.0
 - Add return codes check and handling.
- Version 2.1.1
 - Add DCACHE support.
- Version 2.1.2
 - Add data offset feature to provide support for mirrored (high-speed) memory.
- Version 2.1.3
 - Fix MISRA-2012 issues.
- Version 2.1.4
 - Fix MISRA-2012 issues.
- Version 2.1.5
 - Support EXTENDED data size for all AES, HASH and RNG operations.
 - Support multiple De-Initialization/Initialization of CAAM driver within one POR event.

18.9 Enumeration Type Documentation

18.9.1 anonymous enum

Enumerator

kStatus_CAAM_Again Non-blocking function shall be called again.
kStatus_CAAM_DataOverflow Input data too big.

18.9.2 enum caam_job_ring_t

Enumerator

kCAAM_JobRing0 CAAM Job ring 0.
kCAAM_JobRing1 CAAM Job ring 1.
kCAAM_JobRing2 CAAM Job ring 2.
kCAAM_JobRing3 CAAM Job ring 3.

18.9.3 enum caam_wait_mode_t

Enumerator

kCAAM_Blocking CAAM_Wait blocking mode.

kCAAM_Nonblocking CAAM_Wait non-blocking mode.

18.9.4 enum caam_rng_sample_mode_t

Used by [caam_config_t](#).

Enumerator

kCAAM_RNG_SampleModeVonNeumann Use von Neumann data in both Entropy shifter and Statistical Checker.

kCAAM_RNG_SampleModeRaw Use raw data into both Entropy shifter and Statistical Checker.

kCAAM_RNG_SampleModeVonNeumannRaw Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

18.9.5 enum caam_rng_ring_osc_div_t

Used by [caam_config_t](#).

Enumerator

kCAAM_RNG_RingOscDiv0 Ring oscillator with no divide.

kCAAM_RNG_RingOscDiv2 Ring oscillator divided-by-2.

kCAAM_RNG_RingOscDiv4 Ring oscillator divided-by-4.

kCAAM_RNG_RingOscDiv8 Ring oscillator divided-by-8.

18.9.6 enum caam_priblob_t

Used by [caam_config_t](#).

Enumerator

kCAAM_PrivSecureBootBlobs Private secure boot software blobs.

kCAAM_PrivProvisioningBlobsType1 Private Provisioning Type 1 blobs.

kCAAM_PrivProvisioningBlobsType2 Private Provisioning Type 2 blobs.

kCAAM_NormalOperationBlobs Normal operation blobs.

18.9.7 enum caam_ext_key_xfr_source_t

Enumerator

kCAAM_ExtKeyXfr_KeyRegisterClass1 The Class 1 Key Register is the source.

kCAAM_ExtKeyXfr_KeyRegisterClass2 The Class 2 Key Register is the source.

kCAAM_ExtKeyXfr_PkhaRamE The PKHA E RAM is the source.

18.10 Function Documentation

18.10.1 status_t CAAM_Init (CAAM_Type * *base*, const caam_config_t * *config*)

This function initializes the CAAM driver, including CAAM's internal RNG.

Parameters

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

Returns

kStatus_Success the CAAM Init has completed with zero termination status word

kStatus_Fail the CAAM Init has completed with non-zero termination status word

18.10.2 status_t CAAM_Deinit (CAAM_Type * *base*)

This function deinitializes the CAAM driver.

Parameters

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

Returns

kStatus_Success the CAAM Deinit has completed with zero termination status word

kStatus_Fail the CAAM Deinit has completed with non-zero termination status word

18.10.3 void CAAM_GetDefaultConfig (caam_config_t * *config*)

This function initializes the CAAM configuration structure to a default value. The default values are as follows. caamConfig->rngSampleMode = kCAAM_RNG_SampleModeVonNeumann; caamConfig->rngRingOscDiv = kCAAM_RNG_RingOscDiv4;

Parameters

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

18.10.4 status_t CAAM_Wait (CAAM_Type * *base*, caam_handle_t * *handle*, uint32_t * *descriptor*, caam_wait_mode_t *mode*)

This function polls CAAM output ring for a specific job.

The CAAM job ring is specified by the jobRing field in the [caam_handle_t](#) structure. The job to be waited is specified by its descriptor address.

This function has two modes, determined by the mode argument. In blocking mode, the function polls the specified jobRing until the descriptor is available in the CAAM output job ring. In non-blocking mode, it polls the output ring once and returns status immediately.

The function can be called from multiple threads or interrupt service routines, as internally it uses global critical section (global interrupt disable enable) to protect its operation against concurrent accesses. The global interrupt is disabled only when the descriptor is found in the output ring, for a very short time, to remove the descriptor from the output ring safely.

Parameters

<i>base</i>	CAAM peripheral base address
<i>handle</i>	Data structure with CAAM jobRing used for this request
<i>descriptor</i>	
<i>mode</i>	Blocking and non-blocking mode. Zero is blocking. Non-zero is non-blocking.

Returns

kStatus_Success the CAAM job has completed with zero job termination status word
 kStatus_Fail the CAAM job has completed with non-zero job termination status word
 kStatus_Again In non-blocking mode, the job is not ready in the CAAM Output Ring

18.10.5 status_t CAAM_ExternalKeyTransfer (CAAM_Type * *base*, caam_handle_t * *handle*, caam_ext_key_xfr_source_t *keySource*, size_t *keySize*)

This function loads the given key source to an CAAM external destination via a private interface, such as Inline Encryption Engine IEE Private Key bus.

The CAAM job ring is specified by the jobRing field in the [caam_handle_t](#) structure.

This function is blocking.

Parameters

<i>base</i>	CAAM peripheral base address
<i>handle</i>	Data structure with CAAM jobRing used for this request.
<i>keySource</i>	The source from which the key will be obtained.
<i>keySize</i>	Size of the key in bytes.

Returns

kStatus_Success the CAAM job has completed with zero job termination status word

kStatus_Fail the CAAM job has completed with non-zero job termination status word

18.11 CAAM Blocking APIs

18.11.1 Overview

This section describes the programming interface of the CAAM Synchronous Blocking functions

Modules

- [CAAM AES driver](#)
- [CAAM DES driver](#)
- [CAAM HASH driver](#)
- [CAAM PKHA driver](#)
- [CAAM RNG driver](#)

18.11.2 CAAM RNG driver

18.11.2.1 Overview

This section describes the programming interface of the CAAM RNG driver.

Data Structures

- struct `caam_rng_config_t`
CAAM RNG configuration. [More...](#)

Typedefs

- typedef uint32_t `caam_rng_generic256_t` [256/sizeof(uint32_t)]
256-bit value used as optional additional entropy input

Enumerations

- enum `caam_rng_state_handle_t` {
 `kCAAM_RngStateHandle0` = 0u,
 `kCAAM_RngStateHandle1` = 1u }
CAAM RNG state handle.
- enum `caam_rng_random_type_t` {
 `kCAAM_RngDataAny` = 0u,
 `kCAAM_RngDataOddParity` = 1u,
 `kCAAM_RngDataNonZero` = 2u }
Type of random data to generate.

Functions

- `status_t CAAM_RNG_GetDefaultConfig` (`caam_rng_config_t` *config)
Initializes user configuration structure to default.
- `status_t CAAM_RNG_Init` (`CAAM_Type` *base, `caam_handle_t` *handle, `caam_rng_state_handle_t` stateHandle, const `caam_rng_config_t` *config)
Instantiate the CAAM RNG state handle.
- `status_t CAAM_RNG_Deinit` (`CAAM_Type` *base, `caam_handle_t` *handle, `caam_rng_state_handle_t` stateHandle)
Uninstantiate the CAAM RNG state handle.
- `status_t CAAM_RNG_GenerateSecureKey` (`CAAM_Type` *base, `caam_handle_t` *handle, `caam_rng_generic256_t` additionalEntropy)
Generate Secure Key.
- `status_t CAAM_RNG_Reseed` (`CAAM_Type` *base, `caam_handle_t` *handle, `caam_rng_state_handle_t` stateHandle, `caam_rng_generic256_t` additionalEntropy)
Reseed the CAAM RNG state handle.

- `status_t CAAM_RNG_GetRandomData` (CAAM_Type *base, `caam_handle_t` *handle, `caam_rng_state_handle_t` stateHandle, void *data, size_t dataSize, `caam_rng_random_type_t` dataType, `caam_rng_generic256_t` additionalEntropy)
Get random data.

18.11.2.2 Data Structure Documentation

18.11.2.2.1 struct caam_rng_config_t

Data Fields

- `uint32_t autoReseedInterval`
Automatic reseed interval.
- `caam_rng_generic256_t *personalString`
NULL or pointer to optional personalization string.

Field Documentation

(1) `uint32_t caam_rng_config_t::autoReseedInterval`

If set to zero, CAAM RNG will use hardware default interval of 10.000.000 generate requests.

18.11.2.3 Enumeration Type Documentation

18.11.2.3.1 enum caam_rng_state_handle_t

Enumerator

kCAAM_RngStateHandle0 CAAM RNG state handle 0.
kCAAM_RngStateHandle1 CAAM RNG state handle 1.

18.11.2.3.2 enum caam_rng_random_type_t

Enumerator

kCAAM_RngDataAny CAAM RNG any random data bytes.
kCAAM_RngDataOddParity CAAM RNG odd parity random data bytes.
kCAAM_RngDataNonZero CAAM RNG non zero random data bytes.

18.11.2.4 Function Documentation

18.11.2.4.1 `status_t CAAM_RNG_GetDefaultConfig` (`caam_rng_config_t` * *config*)

This function initializes the configure structure to default value. the default value are:


```

*   config->autoReseedInterval = 0;
*   config->personalString = NULL;
*

```

Parameters

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

Returns

status of the request

18.11.2.4.2 **status_t CAAM_RNG_Init (CAAM_Type * *base*, caam_handle_t * *handle*, caam_rng_state_handle_t *stateHandle*, const caam_rng_config_t * *config*)**

This function instantiates CAAM RNG state handle. The function is blocking and returns after CAAM has processed the request.

Parameters

<i>base</i>	CAAM peripheral base address
<i>handle</i>	CAAM jobRing used for this request
<i>stateHandle</i>	RNG state handle to instantiate
<i>config</i>	Pointer to configuration structure.

Returns

Status of the request

18.11.2.4.3 **status_t CAAM_RNG_Deinit (CAAM_Type * *base*, caam_handle_t * *handle*, caam_rng_state_handle_t *stateHandle*)**

This function uninstantiates CAAM RNG state handle. The function is blocking and returns after CAAM has processed the request.

Parameters

<i>base</i>	CAAM peripheral base address
<i>handle</i>	jobRing used for this request.
<i>stateHandle</i>	RNG state handle to uninstantiate

Returns

Status of the request

18.11.2.4.4 **status_t CAAM_RNG_GenerateSecureKey (CAAM_Type * *base*, caam_handle_t * *handle*, caam_rng_generic256_t *additionalEntropy*)**

This function generates random data writes it to Secure Key registers. The function is blocking and returns after CAAM has processed the request. RNG state handle 0 is always used.

Parameters

<i>base</i>	CAAM peripheral base address
<i>handle</i>	jobRing used for this request
<i>additional-Entropy</i>	NULL or Pointer to optional 256-bit additional entropy.

Returns

Status of the request

18.11.2.4.5 **status_t CAAM_RNG_Reseed (CAAM_Type * *base*, caam_handle_t * *handle*, caam_rng_state_handle_t *stateHandle*, caam_rng_generic256_t *additionalEntropy*)**

This function reseeds the CAAM RNG state handle. For a state handle in nondeterministic mode, the DRNG is seeded with 384 bits of entropy from the TRNG and an optional 256-bit additional input from the descriptor via the Class 1 Context Register.

The function is blocking and returns after CAAM has processed the request.

Parameters

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

<i>handle</i>	jobRing used for this request
<i>stateHandle</i>	RNG state handle to reseed
<i>additional-Entropy</i>	NULL or Pointer to optional 256-bit additional entropy.

Returns

Status of the request

18.11.2.4.6 `status_t CAAM_RNG_GetRandomData (CAAM_Type * base, caam_handle_t * handle, caam_rng_state_handle_t stateHandle, void * data, size_t dataSize, caam_rng_random_type_t dataType, caam_rng_generic256_t additionalEntropy)`

This function gets random data from CAAM RNG.

The function is blocking and returns after CAAM has generated the requested data or an error occurred.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	jobRing used for this request
	<i>stateHandle</i>	RNG state handle used to generate random data
out	<i>data</i>	Pointer address used to store random data
	<i>dataSize</i>	Size of the buffer pointed by the data parameter
	<i>dataType</i>	Type of random data to be generated
	<i>additional-Entropy</i>	NULL or Pointer to optional 256-bit additional entropy.

Returns

Status of the request

18.11.3 CAAM DES driver

18.11.3.1 Overview

This section describes the programming interface of the CAAM DES driver.

Macros

- #define `CAAM_DES_KEY_SIZE` 8U
CAAM DES key size - 64 bits.
- #define `CAAM_DES_IV_SIZE` 8
CAAM DES IV size - 8 bytes.

Functions

- `status_t CAAM_DES_EncryptEcb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key[`CAAM_DES_KEY_SIZE`])
Encrypts DES using ECB block mode.
- `status_t CAAM_DES_DecryptEcb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key[`CAAM_DES_KEY_SIZE`])
Decrypts DES using ECB block mode.
- `status_t CAAM_DES_EncryptCbc` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[`CAAM_DES_IV_SIZE`], const uint8_t key[`CAAM_DES_KEY_SIZE`])
Encrypts DES using CBC block mode.
- `status_t CAAM_DES_DecryptCbc` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[`CAAM_DES_IV_SIZE`], const uint8_t key[`CAAM_DES_KEY_SIZE`])
Decrypts DES using CBC block mode.
- `status_t CAAM_DES_EncryptCfb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[`CAAM_DES_IV_SIZE`], const uint8_t key[`CAAM_DES_KEY_SIZE`])
Encrypts DES using CFB block mode.
- `status_t CAAM_DES_DecryptCfb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[`CAAM_DES_IV_SIZE`], const uint8_t key[`CAAM_DES_KEY_SIZE`])
Decrypts DES using CFB block mode.
- `status_t CAAM_DES_EncryptOfb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[`CAAM_DES_IV_SIZE`], const uint8_t key[`CAAM_DES_KEY_SIZE`])
Encrypts DES using OFB block mode.
- `status_t CAAM_DES_DecryptOfb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[`CAAM_DES_IV_SIZE`], const uint8_t key[`CAAM_DES_KEY_SIZE`])
Decrypts DES using OFB block mode.
- `status_t CAAM_DES2_EncryptEcb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key1[`CAAM_DES_KEY_SIZE`], const

uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using ECB block mode with two keys.

- **status_t CAAM_DES2_DecryptEcb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using ECB block mode with two keys.

- **status_t CAAM_DES2_EncryptCbc** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using CBC block mode with two keys.

- **status_t CAAM_DES2_DecryptCbc** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using CBC block mode with two keys.

- **status_t CAAM_DES2_EncryptCfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using CFB block mode with two keys.

- **status_t CAAM_DES2_DecryptCfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using CFB block mode with two keys.

- **status_t CAAM_DES2_EncryptOfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using OFB block mode with two keys.

- **status_t CAAM_DES2_DecryptOfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using OFB block mode with two keys.

- **status_t CAAM_DES3_EncryptEcb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Encrypts triple DES using ECB block mode with three keys.

- **status_t CAAM_DES3_DecryptEcb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Decrypts triple DES using ECB block mode with three keys.

- **status_t CAAM_DES3_EncryptCbc** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Encrypts triple DES using CBC block mode with three keys.

- **status_t CAAM_DES3_DecryptCbc** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Decrypts triple DES using CBC block mode with three keys.

- **status_t CAAM_DES3_EncryptCfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])
Encrypts triple DES using CFB block mode with three keys.
- **status_t CAAM_DES3_DecryptCfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])
Decrypts triple DES using CFB block mode with three keys.
- **status_t CAAM_DES3_EncryptOfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])
Encrypts triple DES using OFB block mode with three keys.
- **status_t CAAM_DES3_DecryptOfb** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])
Decrypts triple DES using OFB block mode with three keys.

18.11.3.2 Macro Definition Documentation

18.11.3.2.1 #define CAAM_DES_KEY_SIZE 8U

18.11.3.3 Function Documentation

18.11.3.3.1 status_t CAAM_DES_EncryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t key[CAAM_DES_KEY_SIZE])

Encrypts DES using ECB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext

	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.2 `status_t CAAM_DES_DecryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using ECB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.3 `status_t CAAM_DES_EncryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using CBC block mode.

Parameters

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

	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.4 `status_t CAAM_DES_DecryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using CBC block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.5 `status_t CAAM_DES_EncryptCfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using CFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for encryption
out	<i>ciphertext</i>	Output ciphertext

Returns

Status from encrypt/decrypt operation

18.11.3.3.6 `status_t CAAM_DES_DecryptCfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using CFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.7 `status_t CAAM_DES_EncryptOfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using OFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.8 `status_t CAAM_DES_DecryptOfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using OFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

18.11.3.3.9 `status_t CAAM_DES2_EncryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.10 `status_t CAAM_DES2_DecryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.11 `status_t CAAM_DES2_EncryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.12 `status_t CAAM_DES2_DecryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.

	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.13 `status_t CAAM_DES2_EncryptCfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.14 `status_t CAAM_DES2_DecryptCfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.15 `status_t CAAM_DES2_EncryptOfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.16 `status_t CAAM_DES2_DecryptOfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.17 `status_t CAAM_DES3_EncryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using ECB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.18 `status_t CAAM_DES3_DecryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using ECB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.19 `status_t CAAM_DES3_EncryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CBC block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

	<i>key3</i>	Third input key for key bundle
--	-------------	--------------------------------

Returns

Status from encrypt/decrypt operation

18.11.3.3.20 `status_t CAAM_DES3_DecryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CBC block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.21 `status_t CAAM_DES3_EncryptCfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.22 `status_t CAAM_DES3_DecryptCfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.23 `status_t CAAM_DES3_EncryptOfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using OFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.3.3.24 `status_t CAAM_DES3_DecryptOfb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using OFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.

	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

18.11.4 CAAM AES driver

18.11.4.1 Overview

This section describes the programming interface of the CAAM AES driver.

Macros

- `#define CAAM_AES_BLOCK_SIZE 16`
AES block size in bytes.

Functions

- `status_t CAAM_AES_EncryptEcb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *key, size_t keySize)
Encrypts AES using the ECB block mode.
- `status_t CAAM_AES_DecryptEcb` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *key, size_t keySize)
Decrypts AES using ECB block mode.
- `status_t CAAM_AES_EncryptCbc` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_AES_BLOCK_SIZE], const uint8_t *key, size_t keySize)
Encrypts AES using CBC block mode.
- `status_t CAAM_AES_DecryptCbc` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_AES_BLOCK_SIZE], const uint8_t *key, size_t keySize)
Decrypts AES using CBC block mode.
- `status_t CAAM_AES_CryptCtr` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *input, uint8_t *output, size_t size, uint8_t counter[CAAM_AES_BLOCK_SIZE], const uint8_t *key, size_t keySize, uint8_t counterlast[CAAM_AES_BLOCK_SIZE], size_t *szLeft)
Encrypts or decrypts AES using CTR block mode.
- `status_t CAAM_AES_EncryptTagCcm` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, uint8_t *tag, size_t tagSize)
Encrypts AES and tags using CCM block mode.
- `status_t CAAM_AES_DecryptTagCcm` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, const uint8_t *tag, size_t tagSize)
Decrypts AES and authenticates using CCM block mode.
- `status_t CAAM_AES_EncryptTagGcm` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, uint8_t *tag, size_t tagSize)
Encrypts AES and tags using GCM block mode.
- `status_t CAAM_AES_DecryptTagGcm` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, const uint8_t *tag, size_t tagSize)

Decrypts AES and authenticates using GCM block mode.

18.11.4.2 Function Documentation

18.11.4.2.1 `status_t CAAM_AES_EncryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * key, size_t keySize)`

Encrypts AES using the ECB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

18.11.4.2.2 `status_t CAAM_AES_DecryptEcb (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * key, size_t keySize)`

Decrypts AES using ECB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input cipher text to decrypt

out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key.
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from decrypt operation

18.11.4.2.3 `status_t CAAM_AES_EncryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_AES_BLOCK_SIZE], const uint8_t * key, size_t keySize)`

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

18.11.4.2.4 `status_t CAAM_AES_DecryptCbc (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_AES_BLOCK_SIZE], const uint8_t * key, size_t keySize)`

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from decrypt operation

18.11.4.2.5 `status_t CAAM_AES_CryptCtr (CAAM_Type * base, caam_handle_t * handle, const uint8_t * input, uint8_t * output, size_t size, uint8_t counter[CAAM_AES_BLOCK_SIZE], const uint8_t * key, size_t keySize, uint8_t counterlast[CAAM_AES_BLOCK_SIZE], size_t * szLeft)`

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>input</i>	Input data for CTR block mode
out	<i>output</i>	Output data for CTR block mode
	<i>size</i>	Size of input and output data in bytes
in, out	<i>counter</i>	Input counter (updates on return)
	<i>key</i>	Input key to use for forward AES cipher

	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>counterlast</i>	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
out	<i>szLeft</i>	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from encrypt operation

18.11.4.2.6 `status_t CAAM_AES_EncryptTagCcm (CAAM_Type * base, caam_handle_t * handle, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, uint8_t * tag, size_t tagSize)`

Encrypts AES and optionally tags using CCM block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text.
	<i>size</i>	Size of input and output data in bytes. Zero means authentication only.
	<i>iv</i>	Nonce
	<i>ivSize</i>	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	<i>aad</i>	Input additional authentication data. Can be NULL if aadSize is zero.
	<i>aadSize</i>	Input size in bytes of AAD. Zero means data mode only (authentication skipped).
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>tag</i>	Generated output tag. Set to NULL to skip tag processing.

	<i>tagSize</i>	Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.
--	----------------	--

Returns

Status from encrypt operation

18.11.4.2.7 `status_t CAAM_AES_DecryptTagCcm (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, const uint8_t * tag, size_t tagSize)`

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text.
	<i>size</i>	Size of input and output data in bytes. Zero means authentication data only.
	<i>iv</i>	Nonce
	<i>ivSize</i>	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	<i>aad</i>	Input additional authentication data. Can be NULL if aadSize is zero.
	<i>aadSize</i>	Input size in bytes of AAD. Zero means data mode only (authentication data skipped).
	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>tag</i>	Received tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from decrypt operation

18.11.4.2.8 `status_t CAAM_AES_EncryptTagGcm (CAAM_Type * base, caam_handle_t * handle,
const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv, size_t
ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize,
uint8_t * tag, size_t tagSize)`

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text.
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector
	<i>ivSize</i>	Size of the IV
	<i>aad</i>	Input additional authentication data
	<i>aadSize</i>	Input size in bytes of AAD
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>tag</i>	Output hash tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from encrypt operation

18.11.4.2.9 `status_t CAAM_AES_DecryptTagGcm (CAAM_Type * base, caam_handle_t * handle, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, const uint8_t * tag, size_t tagSize)`

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>ciphertext</i>	Input cipher text to decrypt

out	<i>plaintext</i>	Output plain text.
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector
	<i>ivSize</i>	Size of the IV
	<i>aad</i>	Input additional authentication data
	<i>aadSize</i>	Input size in bytes of AAD
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>tag</i>	Input hash tag to compare. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from decrypt operation

18.11.5 CAAM HASH driver

18.11.5.1 Overview

This section describes the programming interface of the CAAM HASH driver.

Macros

- #define `CAAM_SHA_BLOCK_SIZE` 128U
CAAM HASH Context size.
- #define `CAAM_HASH_BLOCK_SIZE` `CAAM_SHA_BLOCK_SIZE`
CAAM hash block size.
- #define `CAAM_HASH_CTX_SIZE` 58
CAAM HASH Context size.

Typedefs

- typedef uint32_t `caam_hash_ctx_t` [`CAAM_HASH_CTX_SIZE`]
Storage type used to save hash context.

Enumerations

- enum `caam_hash_algo_t` {
 `kCAAM_XcbcMac` = 0,
 `kCAAM_Cmac`,
 `kCAAM_Sha1`,
 `kCAAM_Sha224`,
 `kCAAM_Sha256`,
 `kCAAM_Sha384`,
 `kCAAM_Sha512` }
Supported cryptographic block cipher functions for HASH creation.

Functions

- `status_t CAAM_HASH_Init` (`CAAM_Type` *base, `caam_handle_t` *handle, `caam_hash_ctx_t` *ctx, `caam_hash_algo_t` algo, const uint8_t *key, size_t keySize)
Initialize HASH context.
- `status_t CAAM_HASH_Update` (`caam_hash_ctx_t` *ctx, const uint8_t *input, size_t inputSize)
Add data to current HASH.
- `status_t CAAM_HASH_Finish` (`caam_hash_ctx_t` *ctx, uint8_t *output, size_t *outputSize)
Finalize hashing.
- `status_t CAAM_HASH` (`CAAM_Type` *base, `caam_handle_t` *handle, `caam_hash_algo_t` algo, const uint8_t *input, size_t inputSize, const uint8_t *key, size_t keySize, uint8_t *output, size_t *outputSize)
Create HASH on given data.

18.11.5.2 Macro Definition Documentation

18.11.5.2.1 #define CAAM_SHA_BLOCK_SIZE 128U

up to SHA-512 block size

18.11.5.2.2 #define CAAM_HASH_CTX_SIZE 58

18.11.5.3 Typedef Documentation

18.11.5.3.1 typedef uint32_t caam_hash_ctx_t[CAAM_HASH_CTX_SIZE]

18.11.5.4 Enumeration Type Documentation

18.11.5.4.1 enum caam_hash_algo_t

Enumerator

kCAAM_XcbcMac XCBC-MAC (AES engine)
kCAAM_Cmac CMAC (AES engine)
kCAAM_Sha1 SHA_1 (MDHA engine)
kCAAM_Sha224 SHA_224 (MDHA engine)
kCAAM_Sha256 SHA_256 (MDHA engine)
kCAAM_Sha384 SHA_384 (MDHA engine)
kCAAM_Sha512 SHA_512 (MDHA engine)

18.11.5.5 Function Documentation

18.11.5.5.1 status_t CAAM_HASH_Init (CAAM_Type * *base*, caam_handle_t * *handle*, caam_hash_ctx_t * *ctx*, caam_hash_algo_t *algo*, const uint8_t * *key*, size_t *keySize*)

This function initializes the HASH. Key shall be supplied if the underlying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

This functions is used to initialize the context for both blocking and non-blocking CAAM_HASH API. For blocking CAAM HASH API, the HASH context contains all information required for context switch, such as running hash or MAC. For non-blocking CAAM HASH API, the HASH context is used to hold SGT. Therefore, the HASH context cannot be shared between blocking and non-blocking HASH API. With one HASH context, either use only blocking HASH API or only non-blocking HASH API.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request.
out	<i>ctx</i>	Output hash context
	<i>algo</i>	Underlying algorithm to use for hash computation.
	<i>key</i>	Input key (NULL if underlying algorithm is SHA)
	<i>keySize</i>	Size of input key in bytes

Returns

Status of initialization

18.11.5.5.2 **status_t CAAM_HASH_Update (caam_hash_ctx_t * *ctx*, const uint8_t * *input*, size_t *inputSize*)**

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed. The functions blocks. If it returns kStatus_Success, the running hash or mac has been updated (CAAM has processed the input data), so the memory at input pointer can be released back to system. The context is updated with the running hash or mac and with all necessary information to support possible context switch.

Parameters

in, out	<i>ctx</i>	HASH context
	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes

Returns

Status of the hash update operation

18.11.5.5.3 **status_t CAAM_HASH_Finish (caam_hash_ctx_t * *ctx*, uint8_t * *output*, size_t * *outputSize*)**

Outputs the final hash (computed by [CAAM_HASH_Update\(\)](#)) and erases the context.

Parameters

<i>in, out</i>	<i>ctx</i>	Input hash context
<i>out</i>	<i>output</i>	Output hash data
<i>out</i>	<i>outputSize</i>	Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

18.11.5.5.4 `status_t CAAM_HASH (CAAM_Type * base, caam_handle_t * handle,
caam_hash_algo_t algo, const uint8_t * input, size_t inputSize, const uint8_t * key,
size_t keySize, uint8_t * output, size_t * outputSize)`

Perform the full keyed XCBC-MAC/CMAC or SHA in one function call.

Key shall be supplied if the underlying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

The function is blocking.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request.
	<i>algo</i>	Underlying algorithm to use for hash computation.
	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes
	<i>key</i>	Input key (NULL if underlying algorithm is SHA)
	<i>keySize</i>	Size of input key in bytes
<i>out</i>	<i>output</i>	Output hash data
<i>out</i>	<i>outputSize</i>	Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

18.11.6 CAAM PKHA driver

18.11.6.1 Overview

This section describes the programming interface of the CAAM PKHA driver.

Data Structures

- struct `caam_pkha_ecc_point_t`
PKHA ECC point structure. [More...](#)

Enumerations

- enum `caam_pkha_timing_t` {
 `kCAAM_PKHA_NoTimingEqualized` = 0U,
 `kCAAM_PKHA_TimingEqualized` = 1U }
 Use of timing equalized version of a PKHA function.
- enum `caam_pkha_f2m_t` {
 `kCAAM_PKHA_IntegerArith` = 0U,
 `kCAAM_PKHA_F2mArith` = 1U }
 Integer vs binary polynomial arithmetic selection.
- enum `caam_pkha_montgomery_form_t` {
 `kCAAM_PKHA_NormalValue` = 0U,
 `kCAAM_PKHA_MontgomeryFormat` = 1U }
 Montgomery or normal PKHA input format.

Functions

- `status_t CAAM_PKHA_NormalToMontgomery` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *N, size_t sizeN, uint8_t *A, size_t *sizeA, uint8_t *B, size_t *sizeB, uint8_t *R2, size_t *sizeR2, `caam_pkha_timing_t` equalTime, `caam_pkha_f2m_t` arithType)
 Converts from integer to Montgomery format.
- `status_t CAAM_PKHA_MontgomeryToNormal` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *N, size_t sizeN, uint8_t *A, size_t *sizeA, uint8_t *B, size_t *sizeB, `caam_pkha_timing_t` equalTime, `caam_pkha_f2m_t` arithType)
 Converts from Montgomery format to int.
- `status_t CAAM_PKHA_ModAdd` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *A, size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, `caam_pkha_f2m_t` arithType)
 Performs modular addition - (A + B) mod N.
- `status_t CAAM_PKHA_ModSub1` (CAAM_Type *base, `caam_handle_t` *handle, const uint8_t *A, size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize)
 Performs modular subtraction - (A - B) mod N.

- **status_t CAAM_PKHA_ModSub2** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize)
Performs modular subtraction - $(B - A) \bmod N$.
- **status_t CAAM_PKHA_ModMul** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, caam_pkha_f2m_t arithType, caam_pkha_montgomery_form_t montIn, caam_pkha_montgomery_form_t montOut, caam_pkha_timing_t equalTime)
Performs modular multiplication - $(A \times B) \bmod N$.
- **status_t CAAM_PKHA_ModExp** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *N, size_t sizeN, const uint8_t *E, size_t sizeE, uint8_t *result, size_t *resultSize, caam_pkha_f2m_t arithType, caam_pkha_montgomery_form_t montIn, caam_pkha_timing_t equalTime)
Performs modular exponentiation - $(A^E) \bmod N$.
- **status_t CAAM_PKHA_ModRed** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, caam_pkha_f2m_t arithType)
Performs modular reduction - $(A) \bmod N$.
- **status_t CAAM_PKHA_ModInv** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, caam_pkha_f2m_t arithType)
Performs modular inversion - $(A^{-1}) \bmod N$.
- **status_t CAAM_PKHA_ModR2** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, caam_pkha_f2m_t arithType)
Computes integer Montgomery factor $R^2 \bmod N$.
- **status_t CAAM_PKHA_ModGcd** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *N, size_t sizeN, uint8_t *result, size_t *resultSize, caam_pkha_f2m_t arithType)
Calculates the greatest common divisor - $GCD(A, N)$.
- **status_t CAAM_PKHA_PrimalityTest** (CAAM_Type *base, caam_handle_t *handle, const uint8_t *A, size_t sizeA, const uint8_t *B, size_t sizeB, const uint8_t *N, size_t sizeN, bool *res)
Executes Miller-Rabin primality test.
- **status_t CAAM_PKHA_ECC_PointAdd** (CAAM_Type *base, caam_handle_t *handle, const caam_pkha_ecc_point_t *A, const caam_pkha_ecc_point_t *B, const uint8_t *N, const uint8_t *R2modN, const uint8_t *aCurveParam, const uint8_t *bCurveParam, size_t size, caam_pkha_f2m_t arithType, caam_pkha_ecc_point_t *result)
Adds elliptic curve points - $A + B$.
- **status_t CAAM_PKHA_ECC_PointDouble** (CAAM_Type *base, caam_handle_t *handle, const caam_pkha_ecc_point_t *B, const uint8_t *N, const uint8_t *aCurveParam, const uint8_t *bCurveParam, size_t size, caam_pkha_f2m_t arithType, caam_pkha_ecc_point_t *result)
Doubles elliptic curve points - $B + B$.
- **status_t CAAM_PKHA_ECC_PointMul** (CAAM_Type *base, caam_handle_t *handle, const caam_pkha_ecc_point_t *A, const uint8_t *E, size_t sizeE, const uint8_t *N, const uint8_t *R2modN, const uint8_t *aCurveParam, const uint8_t *bCurveParam, size_t size, caam_pkha_timing_t equalTime, caam_pkha_f2m_t arithType, caam_pkha_ecc_point_t *result)
Multiplies an elliptic curve point by a scalar - $E \times (A0, A1)$.

18.11.6.2 Data Structure Documentation

18.11.6.2.1 struct caam_pkha_ecc_point_t

Data Fields

- uint8_t * *X*
X coordinate (affine)
- uint8_t * *Y*
Y coordinate (affine)

18.11.6.3 Enumeration Type Documentation

18.11.6.3.1 enum caam_pkha_timing_t

Enumerator

kCAAM_PKHA_NoTimingEqualized Normal version of a PKHA operation.

kCAAM_PKHA_TimingEqualized Timing-equalized version of a PKHA operation.

18.11.6.3.2 enum caam_pkha_f2m_t

Enumerator

kCAAM_PKHA_IntegerArith Use integer arithmetic.

kCAAM_PKHA_F2mArith Use binary polynomial arithmetic.

18.11.6.3.3 enum caam_pkha_montgomery_form_t

Enumerator

kCAAM_PKHA_NormalValue PKHA number is normal integer.

kCAAM_PKHA_MontgomeryFormat PKHA number is in montgomery format.

18.11.6.4 Function Documentation

18.11.6.4.1 status_t CAAM_PKHA_NormalToMontgomery (CAAM_Type * *base*, caam_handle_t * *handle*, const uint8_t * *N*, size_t *sizeN*, uint8_t * *A*, size_t * *sizeA*, uint8_t * *B*, size_t * *sizeB*, uint8_t * *R2*, size_t * *sizeR2*, caam_pkha_timing_t *equalTime*, caam_pkha_f2m_t *arithType*)

This function computes $R2 \bmod N$ and optionally converts A or B into Montgomery format of A or B.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>N</i>	modulus
	<i>sizeN</i>	size of N in bytes
in, out	<i>A</i>	The first input in non-Montgomery format. Output Montgomery format of the first input.
in, out	<i>sizeA</i>	pointer to size variable. On input it holds size of input A in bytes. On output it holds size of Montgomery format of A in bytes.
in, out	<i>B</i>	Second input in non-Montgomery format. Output Montgomery format of the second input.
in, out	<i>sizeB</i>	pointer to size variable. On input it holds size of input B in bytes. On output it holds size of Montgomery format of B in bytes.
out	<i>R2</i>	Output Montgomery factor R2 mod N.
out	<i>sizeR2</i>	pointer to size variable. On output it holds size of Montgomery factor R2 mod N in bytes.
	<i>equalTime</i>	Run the function time equalized or no timing equalization.
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.2 `status_t CAAM_PKHA_MontgomeryToNormal (CAAM_Type * base, caam_handle_t * handle, const uint8_t * N, size_t sizeN, uint8_t * A, size_t * sizeA, uint8_t * B, size_t * sizeB, caam_pkha_timing_t equalTime, caam_pkha_f2m_t arithType)`

This function converts Montgomery format of A or B into int A or B.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.

	<i>N</i>	modulus.
	<i>sizeN</i>	size of N modulus in bytes.
<i>in, out</i>	<i>A</i>	Input first number in Montgomery format. Output is non-Montgomery format.
<i>in, out</i>	<i>sizeA</i>	pointer to size variable. On input it holds size of the input A in bytes. On output it holds size of non-Montgomery A in bytes.
<i>in, out</i>	<i>B</i>	Input first number in Montgomery format. Output is non-Montgomery format.
<i>in, out</i>	<i>sizeB</i>	pointer to size variable. On input it holds size of the input B in bytes. On output it holds size of non-Montgomery B in bytes.
	<i>equalTime</i>	Run the function time equalized or no timing equalization.
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.3 `status_t CAAM_PKHA_ModAdd (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * B, size_t sizeB, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize, caam_pkha_f2m_t arithType)`

This function performs modular addition of $(A + B) \bmod N$, with either integer or binary polynomial (F2m) inputs. In the F2m form, this function is equivalent to a bitwise XOR and it is functionally the same as subtraction.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes

	<i>N</i>	modulus.
	<i>sizeN</i>	Size of N in bytes.
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.4 `status_t CAAM_PKHA_ModSub1 (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * B, size_t sizeB, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize)`

This function performs modular subtraction of (A - B) mod N with integer inputs.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes

Returns

Operation status.

18.11.6.4.5 `status_t CAAM_PKHA_ModSub2 (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * B, size_t sizeB, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize)`

This function performs modular subtraction of (B - A) mod N, with integer inputs.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes

Returns

Operation status.

18.11.6.4.6 `status_t CAAM_PKHA_ModMul (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * B, size_t sizeB, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize, caam_pkha_f2m_t arithType, caam_pkha_montgomery_form_t montIn, caam_pkha_montgomery_form_t montOut, caam_pkha_timing_t equalTime)`

This function performs modular multiplication with either integer or binary polynomial (F2m) inputs. It can optionally specify whether inputs and/or outputs will be in Montgomery form or not.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	second addend (integer or binary polynomial)

	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	modulus.
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
	<i>montIn</i>	Format of inputs
	<i>montOut</i>	Format of output
	<i>equalTime</i>	Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

Returns

Operation status.

18.11.6.4.7 `status_t CAAM_PKHA_ModExp (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * N, size_t sizeN, const uint8_t * E, size_t sizeE, uint8_t * result, size_t * resultSize, caam_pkha_f2m_t arithType, caam_pkha_montgomery_form_t montIn, caam_pkha_timing_t equalTime)`

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
	<i>E</i>	exponent
	<i>sizeE</i>	Size of E in bytes

out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>montIn</i>	Format of A input (normal or Montgomery)
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
	<i>equalTime</i>	Run the function time equalized or no timing equalization.

Returns

Operation status.

18.11.6.4.8 `status_t CAAM_PKHA_ModRed (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize, caam_pkha_f2m_t arithType)`

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.9 `status_t CAAM_PKHA_ModInv (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize, caam_pkha_f2m_t arithType)`

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first addend (integer or binary polynomial)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.10 `status_t CAAM_PKHA_ModR2 (CAAM_Type * base, caam_handle_t * handle, const uint8_t * N, size_t sizeN, uint8_t * result, size_t * resultSize, caam_pkha_f2m_t arithType)`

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>N</i>	modulus
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.11 `status_t CAAM_PKHA_ModGcd (CAAM_Type * base, caam_handle_t * handle,
const uint8_t * A, size_t sizeA, const uint8_t * N, size_t sizeN, uint8_t * result,
size_t * resultSize, caam_pkha_f2m_t arithType)`

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	first value (must be smaller than or equal to N)
	<i>sizeA</i>	Size of A in bytes
	<i>N</i>	second value (must be non-zero)
	<i>sizeN</i>	Size of N in bytes
out	<i>result</i>	Output array to store result of operation
out	<i>resultSize</i>	Output size of operation in bytes
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

18.11.6.4.12 `status_t CAAM_PKHA_PrimalityEngine (CAAM_Type * base, caam_handle_t * handle, const uint8_t * A, size_t sizeA, const uint8_t * B, size_t sizeB, const uint8_t * N, size_t sizeN, bool * res)`

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	initial random seed
	<i>sizeA</i>	Size of A in bytes
	<i>B</i>	number of trial runs
	<i>sizeB</i>	Size of B in bytes
	<i>N</i>	candidate prime integer
	<i>sizeN</i>	Size of N in bytes

out	res	True if the value is likely prime or false otherwise
-----	-----	--

Returns

Operation status.

18.11.6.4.13 `status_t CAAM_PKHA_ECC_PointAdd (CAAM_Type * base, caam_handle_t * handle, const caam_pkha_ecc_point_t * A, const caam_pkha_ecc_point_t * B, const uint8_t * N, const uint8_t * R2modN, const uint8_t * aCurveParam, const uint8_t * bCurveParam, size_t size, caam_pkha_f2m_t arithType, caam_pkha_ecc_point_t * result)`

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	Left-hand point
	<i>B</i>	Right-hand point
	<i>N</i>	Prime modulus of the field
	<i>R2modN</i>	NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from CAAM_PKHA_ModR2() function).
	<i>aCurveParam</i>	A parameter from curve equation
	<i>bCurveParam</i>	B parameter from curve equation (constant)
	<i>size</i>	Size in bytes of curve points and parameters
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
out	<i>result</i>	Result point

Returns

Operation status.

18.11.6.4.14 `status_t CAAM_PKHA_ECC_PointDouble (CAAM_Type * base, caam_handle_t * handle, const caam_pkha_ecc_point_t * B, const uint8_t * N, const uint8_t * aCurveParam, const uint8_t * bCurveParam, size_t size, caam_pkha_f2m_t arithType, caam_pkha_ecc_point_t * result)`

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>B</i>	Point to double
	<i>N</i>	Prime modulus of the field
	<i>aCurveParam</i>	A parameter from curve equation
	<i>bCurveParam</i>	B parameter from curve equation (constant)
	<i>size</i>	Size in bytes of curve points and parameters
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
out	<i>result</i>	Result point

Returns

Operation status.

18.11.6.4.15 `status_t CAAM_PKHA_ECC_PointMul (CAAM_Type * base, caam_handle_t * handle, const caam_pkha_ecc_point_t * A, const uint8_t * E, size_t sizeE, const uint8_t * N, const uint8_t * R2modN, const uint8_t * aCurveParam, const uint8_t * bCurveParam, size_t size, caam_pkha_timing_t equalTime, caam_pkha_f2m_t arithType, caam_pkha_ecc_point_t * result)`

This function performs ECC point multiplication to multiply an ECC point by a scalar integer multiplier over a prime field (Fp) or a binary field (F2m).

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>A</i>	Point as multiplicand
	<i>E</i>	Scalar multiple
	<i>sizeE</i>	The size of E, in bytes
	<i>N</i>	Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.

	<i>R2modN</i>	NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from CAAM_PKHA_ModR2() function).
	<i>aCurveParam</i>	A parameter from curve equation
	<i>bCurveParam</i>	B parameter from curve equation (C parameter for operation over F2m).
	<i>size</i>	Size in bytes of curve points and parameters
	<i>equalTime</i>	Run the function time equalized or no timing equalization.
	<i>arithType</i>	Type of arithmetic to perform (integer or F2m)
out	<i>result</i>	Result point

Returns

Operation status.

18.12 CAAM Non-blocking APIs

18.12.1 Overview

This section describes the programming interface of the CAAM Non Blocking functions

Modules

- [CAAM Non-blocking AES driver](#)
- [CAAM Non-blocking DES driver](#)
- [CAAM Non-blocking HASH driver](#)
- [CAAM Non-blocking RNG driver](#)

18.12.2 CAAM Non-blocking DES driver

18.12.2.1 Overview

This section describes the programming interface of the CAAM Non-blocking DES driver.

Functions

- [status_t CAAM_DES_EncryptEcbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key[CAAM_DES_KEY_SIZE])
Encrypts DES using ECB block mode.
- [status_t CAAM_DES_DecryptEcbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key[CAAM_DES_KEY_SIZE])
Decrypts DES using ECB block mode.
- [status_t CAAM_DES_EncryptCbcNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])
Encrypts DES using CBC block mode.
- [status_t CAAM_DES_DecryptCbcNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])
Decrypts DES using CBC block mode.
- [status_t CAAM_DES_EncryptCfbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])
Encrypts DES using CFB block mode.
- [status_t CAAM_DES_DecryptCfbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])
Decrypts DES using CFB block mode.
- [status_t CAAM_DES_EncryptOfbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])
Encrypts DES using OFB block mode.
- [status_t CAAM_DES_DecryptOfbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])
Decrypts DES using OFB block mode.
- [status_t CAAM_DES2_EncryptEcbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])
Encrypts triple DES using ECB block mode with two keys.
- [status_t CAAM_DES2_DecryptEcbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_cipher_des_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using ECB block mode with two keys.

- `status_t CAAM_DES2_EncryptCbcNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using CBC block mode with two keys.

- `status_t CAAM_DES2_DecryptCbcNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using CBC block mode with two keys.

- `status_t CAAM_DES2_EncryptCfbNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using CFB block mode with two keys.

- `status_t CAAM_DES2_DecryptCfbNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using CFB block mode with two keys.

- `status_t CAAM_DES2_EncryptOfbNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Encrypts triple DES using OFB block mode with two keys.

- `status_t CAAM_DES2_DecryptOfbNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])

Decrypts triple DES using OFB block mode with two keys.

- `status_t CAAM_DES3_EncryptEcbNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Encrypts triple DES using ECB block mode with three keys.

- `status_t CAAM_DES3_DecryptEcbNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Decrypts triple DES using ECB block mode with three keys.

- `status_t CAAM_DES3_EncryptCbcNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle, `caam_desc_cipher_des_t` descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])

Encrypts triple DES using CBC block mode with three keys.

- `status_t CAAM_DES3_DecryptCbcNonBlocking` (CAAM_Type *base, `caam_handle_t` *handle,

`caam_desc_cipher_des_t` descriptor, `const uint8_t *ciphertext`, `uint8_t *plaintext`, `size_t size`, `const uint8_t iv[CAAM_DES_IV_SIZE]`, `const uint8_t key1[CAAM_DES_KEY_SIZE]`, `const uint8_t key2[CAAM_DES_KEY_SIZE]`, `const uint8_t key3[CAAM_DES_KEY_SIZE]`)

Decrypts triple DES using CBC block mode with three keys.

- `status_t CAAM_DES3_EncryptCfbNonBlocking` (`CAAM_Type *base`, `caam_handle_t *handle`, `caam_desc_cipher_des_t` descriptor, `const uint8_t *plaintext`, `uint8_t *ciphertext`, `size_t size`, `const uint8_t iv[CAAM_DES_IV_SIZE]`, `const uint8_t key1[CAAM_DES_KEY_SIZE]`, `const uint8_t key2[CAAM_DES_KEY_SIZE]`, `const uint8_t key3[CAAM_DES_KEY_SIZE]`)

Encrypts triple DES using CFB block mode with three keys.

- `status_t CAAM_DES3_DecryptCfbNonBlocking` (`CAAM_Type *base`, `caam_handle_t *handle`, `caam_desc_cipher_des_t` descriptor, `const uint8_t *ciphertext`, `uint8_t *plaintext`, `size_t size`, `const uint8_t iv[CAAM_DES_IV_SIZE]`, `const uint8_t key1[CAAM_DES_KEY_SIZE]`, `const uint8_t key2[CAAM_DES_KEY_SIZE]`, `const uint8_t key3[CAAM_DES_KEY_SIZE]`)

Decrypts triple DES using CFB block mode with three keys.

- `status_t CAAM_DES3_EncryptOfbNonBlocking` (`CAAM_Type *base`, `caam_handle_t *handle`, `caam_desc_cipher_des_t` descriptor, `const uint8_t *plaintext`, `uint8_t *ciphertext`, `size_t size`, `const uint8_t iv[CAAM_DES_IV_SIZE]`, `const uint8_t key1[CAAM_DES_KEY_SIZE]`, `const uint8_t key2[CAAM_DES_KEY_SIZE]`, `const uint8_t key3[CAAM_DES_KEY_SIZE]`)

Encrypts triple DES using OFB block mode with three keys.

- `status_t CAAM_DES3_DecryptOfbNonBlocking` (`CAAM_Type *base`, `caam_handle_t *handle`, `caam_desc_cipher_des_t` descriptor, `const uint8_t *ciphertext`, `uint8_t *plaintext`, `size_t size`, `const uint8_t iv[CAAM_DES_IV_SIZE]`, `const uint8_t key1[CAAM_DES_KEY_SIZE]`, `const uint8_t key2[CAAM_DES_KEY_SIZE]`, `const uint8_t key3[CAAM_DES_KEY_SIZE]`)

Decrypts triple DES using OFB block mode with three keys.

18.12.2.2 Function Documentation

18.12.2.2.1 `status_t CAAM_DES_EncryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using ECB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt

out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for encryption

Returns

Status from descriptor push

18.12.2.2.2 `status_t CAAM_DES_DecryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using ECB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key</i>	Input key to use for decryption

Returns

Status from descriptor push

18.12.2.2.3 `status_t CAAM_DES_EncryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using CBC block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key</i>	Input key to use for encryption

Returns

Status from descriptor push

18.12.2.2.4 `status_t CAAM_DES_DecryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using CBC block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.

	<i>key</i>	Input key to use for decryption
--	------------	---------------------------------

Returns

Status from descriptor push

18.12.2.2.5 `status_t CAAM_DES_EncryptCfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using CFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for encryption
out	<i>ciphertext</i>	Output ciphertext

Returns

Status from descriptor push

18.12.2.2.6 `status_t CAAM_DES_DecryptCfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using CFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key</i>	Input key to use for decryption

Returns

Status from descriptor push

18.12.2.2.7 `status_t CAAM_DES_EncryptOfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Encrypts DES using OFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.

	<i>key</i>	Input key to use for encryption
--	------------	---------------------------------

Returns

Status from descriptor push

18.12.2.2.8 `status_t CAAM_DES_DecryptOfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key[CAAM_DES_KEY_SIZE])`

Decrypts DES using OFB block mode.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key</i>	Input key to use for decryption

Returns

Status from descriptor push

18.12.2.2.9 `status_t CAAM_DES2_EncryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.10 `status_t CAAM_DES2_DecryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using ECB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.11 `status_t CAAM_DES2_EncryptCbcNonBlocking (CAAM_Type * base,
caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const
uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t
iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t
key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.12 `status_t CAAM_DES2_DecryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CBC block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes

	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.13 `status_t CAAM_DES2_EncryptCfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.14 `status_t CAAM_DES2_DecryptCfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.15 `status_t CAAM_DES2_EncryptOfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.

	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.16 `status_t CAAM_DES2_DecryptOfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using OFB block mode with two keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle

Returns

Status from descriptor push

18.12.2.2.17 `status_t CAAM_DES3_EncryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using ECB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.18 `status_t CAAM_DES3_DecryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using ECB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 8 bytes.
	<i>key1</i>	First input key for key bundle

	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.19 `status_t CAAM_DES3_EncryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CBC block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.20 `status_t CAAM_DES3_DecryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CBC block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.21 `status_t CAAM_DES3_EncryptCfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using CFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.22 `status_t CAAM_DES3_DecryptCfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using CFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input data in bytes
	<i>iv</i>	Input initial block.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.23 `status_t CAAM_DES3_EncryptOfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Encrypts triple DES using OFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>plaintext</i>	Input plaintext to encrypt
out	<i>ciphertext</i>	Output ciphertext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.2.2.24 `status_t CAAM_DES3_DecryptOfbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_cipher_des_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t iv[CAAM_DES_IV_SIZE], const uint8_t key1[CAAM_DES_KEY_SIZE], const uint8_t key2[CAAM_DES_KEY_SIZE], const uint8_t key3[CAAM_DES_KEY_SIZE])`

Decrypts triple DES using OFB block mode with three keys.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	memory for CAAM commands
	<i>ciphertext</i>	Input ciphertext to decrypt
out	<i>plaintext</i>	Output plaintext
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	<i>key1</i>	First input key for key bundle
	<i>key2</i>	Second input key for key bundle
	<i>key3</i>	Third input key for key bundle

Returns

Status from descriptor push

18.12.3 CAAM Non-blocking HASH driver

18.12.3.1 Overview

This section describes the programming interface of the CAAM Non-blocking HASH driver.

Functions

- [status_t CAAM_HASH_UpdateNonBlocking](#) ([caam_hash_ctx_t](#) *ctx, const uint8_t *input, size_t inputSize)
Add input address and size to input data table.
- [status_t CAAM_HASH_FinishNonBlocking](#) ([caam_hash_ctx_t](#) *ctx, [caam_desc_hash_t](#) descriptor, uint8_t *output, size_t *outputSize)
Finalize hashing.
- [status_t CAAM_HASH_NonBlocking](#) ([CAAM_Type](#) *base, [caam_handle_t](#) *handle, [caam_desc_hash_t](#) descriptor, [caam_hash_algo_t](#) algo, const uint8_t *input, size_t inputSize, const uint8_t *key, size_t keySize, uint8_t *output, size_t *outputSize)
Create HASH on given data.

18.12.3.2 Function Documentation

18.12.3.2.1 [status_t CAAM_HASH_UpdateNonBlocking](#) ([caam_hash_ctx_t](#) * ctx, const uint8_t * input, size_t inputSize)

Add data input pointer to a table maintained internally in the context. Each call of this function creates one entry in the table. The entry consists of the input pointer and inputSize. All entries created by one or multiple calls of this function can be processed in one call to [CAAM_HASH_FinishNonBlocking\(\)](#) function. Individual entries can point to non-continuous data in the memory. The processing will occur in the order in which the [CAAM_HASH_UpdateNonBlocking\(\)](#) have been called.

Memory pointers will be later accessed by CAAM (at time of [CAAM_HASH_FinishNonBlocking\(\)](#)), so the memory must stay valid until [CAAM_HASH_FinishNonBlocking\(\)](#) has been called and CAAM completes the processing.

Parameters

in, out	ctx	HASH context
	input	Input data
	inputSize	Size of input data in bytes

Returns

Status of the hash update operation

18.12.3.2.2 `status_t CAAM_HASH_FinishNonBlocking (caam_hash_ctx_t * ctx,
caam_desc_hash_t descriptor, uint8_t * output, size_t * outputSize)`

The actual algorithm is computed with all input data, the memory pointers are accessed by CAAM after the function returns. The input data chunks have been specified by prior calls to [CAAM_HASH_UpdateNonBlocking\(\)](#). The function schedules the request at CAAM, then returns. After a while, when the CAAM completes processing of the input data chunks, the result is written to the `output[]` array, `outputSize` is written and the context is cleared.

Parameters

<code>in, out</code>	<code>ctx</code>	Input hash context
<code>out</code>	<code>descriptor</code>	Memory for the CAAM descriptor.
<code>out</code>	<code>output</code>	Output hash data
<code>out</code>	<code>outputSize</code>	Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

18.12.3.2.3 `status_t CAAM_HASH_NonBlocking (CAAM_Type * base, caam_handle_t * handle,
caam_desc_hash_t descriptor, caam_hash_algo_t algo, const uint8_t * input, size_t
inputSize, const uint8_t * key, size_t keySize, uint8_t * output, size_t * outputSize)`

Perform the full keyed XCBC-MAC/CMAC or SHA in one function call.

Key shall be supplied if the underlying algorithm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlying algorithm is SHA.

For XCBC-MAC, the key length must be 16. For CMAC, the key length can be the AES key lengths supported by AES engine. For MDHA the key length argument is ignored.

The function is non-blocking. The request is scheduled at CAAM.

Parameters

	<code>base</code>	CAAM peripheral base address
	<code>handle</code>	Handle used for this request.
<code>out</code>	<code>descriptor</code>	Memory for the CAAM descriptor.

	<i>algo</i>	Underlying algorithm to use for hash computation.
	<i>input</i>	Input data
	<i>inputSize</i>	Size of input data in bytes
	<i>key</i>	Input key (NULL if underlying algorithm is SHA)
	<i>keySize</i>	Size of input key in bytes
out	<i>output</i>	Output hash data
out	<i>outputSize</i>	Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

18.12.4 CAAM Non-blocking RNG driver

18.12.4.1 Overview

This section describes the programming interface of the CAAM Non-blocking RNG driver.

Functions

- `status_t CAAM_RNG_GetRandomDataNonBlocking` (CAAM_Type *base, caam_handle_t *handle, caam_rng_state_handle_t stateHandle, caam_desc_rng_t descriptor, void *data, size_t dataSize, caam_rng_random_type_t dataType, caam_rng_generic256_t additionalEntropy)
Request random data.

18.12.4.2 Function Documentation

18.12.4.2.1 `status_t CAAM_RNG_GetRandomDataNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_rng_state_handle_t stateHandle, caam_desc_rng_t descriptor, void * data, size_t dataSize, caam_rng_random_type_t dataType, caam_rng_generic256_t additionalEntropy)`

This function schedules the request for random data from CAAM RNG. Memory at memory pointers will be accessed by CAAM shortly after this function returns, according to actual CAAM schedule.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	RNG handle used for this request
	<i>stateHandle</i>	RNG state handle used to generate random data
out	<i>descriptor</i>	memory for CAAM commands
out	<i>data</i>	Pointer address used to store random data
	<i>dataSize</i>	Size of the buffer pointed by the data parameter, in bytes.
	<i>dataType</i>	Type of random data to be generated.
	<i>additional-Entropy</i>	NULL or Pointer to optional 256-bit additional entropy.

Returns

status of the request

18.12.5 CAAM Non-blocking AES driver

18.12.5.1 Overview

This section describes the programming interface of the CAAM Non-blocking AES driver.

Functions

- [status_t CAAM_AES_EncryptEcbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_ecb_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *key, size_t keySize)
Encrypts AES using the ECB block mode.
- [status_t CAAM_AES_DecryptEcbNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_ecb_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *key, size_t keySize)
Decrypts AES using ECB block mode.
- [status_t CAAM_AES_EncryptCbcNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_cbc_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, const uint8_t *key, size_t keySize)
Encrypts AES using CBC block mode.
- [status_t CAAM_AES_DecryptCbcNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_cbc_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, const uint8_t *key, size_t keySize)
Decrypts AES using CBC block mode.
- [status_t CAAM_AES_CryptCtrNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_ctr_t](#) descriptor, const uint8_t *input, uint8_t *output, size_t size, uint8_t *counter, const uint8_t *key, size_t keySize, uint8_t *counterlast, size_t *szLeft)
Encrypts or decrypts AES using CTR block mode.
- [status_t CAAM_AES_EncryptTagCcmNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_ccm_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, uint8_t *tag, size_t tagSize)
Encrypts AES and tags using CCM block mode.
- [status_t CAAM_AES_DecryptTagCcmNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_ccm_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, const uint8_t *tag, size_t tagSize)
Decrypts AES and authenticates using CCM block mode.
- [status_t CAAM_AES_EncryptTagGcmNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_gcm_t](#) descriptor, const uint8_t *plaintext, uint8_t *ciphertext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t keySize, uint8_t *tag, size_t tagSize)
Encrypts AES and tags using GCM block mode.
- [status_t CAAM_AES_DecryptTagGcmNonBlocking](#) (CAAM_Type *base, [caam_handle_t](#) *handle, [caam_desc_aes_gcm_t](#) descriptor, const uint8_t *ciphertext, uint8_t *plaintext, size_t size, const uint8_t *iv, size_t ivSize, const uint8_t *aad, size_t aadSize, const uint8_t *key, size_t

keySize, const uint8_t *tag, size_t tagSize)
Decrypts AES and authenticates using GCM block mode.

18.12.5.2 Function Documentation

18.12.5.2.1 `status_t CAAM_AES_EncryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_ecb_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * key, size_t keySize)`

Puts AES ECB encrypt descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>descriptor</i>	Memory for the CAAM descriptor.
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

18.12.5.2.2 `status_t CAAM_AES_DecryptEcbNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_ecb_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * key, size_t keySize)`

Puts AES ECB decrypt descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.

out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>key</i>	Input key.
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

18.12.5.2.3 `status_t CAAM_AES_EncryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_cbc_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv, const uint8_t * key, size_t keySize)`

Puts AES CBC encrypt descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

18.12.5.2.4 `status_t CAAM_AES_DecryptCbcNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_cbc_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv, const uint8_t * key, size_t keySize)`

Puts AES CBC decrypt descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text
	<i>size</i>	Size of input and output data in bytes. Must be multiple of 16 bytes.
	<i>iv</i>	Input initial vector to combine with the first input block.
	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from job descriptor push

18.12.5.2.5 `status_t CAAM_AES_CryptCtrNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_ctr_t descriptor, const uint8_t * input, uint8_t * output, size_t size, uint8_t * counter, const uint8_t * key, size_t keySize, uint8_t * counterlast, size_t * szLeft)`

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

Puts AES CTR crypt descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>input</i>	Input data for CTR block mode
out	<i>output</i>	Output data for CTR block mode

	<i>size</i>	Size of input and output data in bytes
<i>in, out</i>	<i>counter</i>	Input counter (updates on return)
	<i>key</i>	Input key to use for forward AES cipher
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
<i>out</i>	<i>counterlast</i>	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
<i>out</i>	<i>szLeft</i>	Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used.

Returns

Status from job descriptor push

18.12.5.2.6 `status_t CAAM_AES_EncryptTagCcmNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_ccm_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, uint8_t * tag, size_t tagSize)`

Puts AES CCM encrypt and tag descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
<i>out</i>	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>plaintext</i>	Input plain text to encrypt
<i>out</i>	<i>ciphertext</i>	Output cipher text.
	<i>size</i>	Size of input and output data in bytes. Zero means authentication only.
	<i>iv</i>	Nonce
	<i>ivSize</i>	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	<i>aad</i>	Input additional authentication data. Can be NULL if aadSize is zero.
	<i>aadSize</i>	Input size in bytes of AAD. Zero means data mode only (authentication skipped).

	<i>key</i>	Input key to use for encryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>tag</i>	Generated output tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from job descriptor push

18.12.5.2.7 `status_t CAAM_AES_DecryptTagCcmNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_ccm_t descriptor, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, const uint8_t * tag, size_t tagSize)`

Puts AES CCM decrypt and check tag descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text.
	<i>size</i>	Size of input and output data in bytes. Zero means authentication data only.
	<i>iv</i>	Nonce
	<i>ivSize</i>	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	<i>aad</i>	Input additional authentication data. Can be NULL if aadSize is zero.
	<i>aadSize</i>	Input size in bytes of AAD. Zero means data mode only (authentication data skipped).

	<i>key</i>	Input key to use for decryption
	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>tag</i>	Received tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the received tag to compare with the computed tag, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from job descriptor push

18.12.5.2.8 `status_t CAAM_AES_EncryptTagGcmNonBlocking (CAAM_Type * base, caam_handle_t * handle, caam_desc_aes_gcm_t descriptor, const uint8_t * plaintext, uint8_t * ciphertext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, uint8_t * tag, size_t tagSize)`

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field. Puts AES GCM encrypt and tag descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>plaintext</i>	Input plain text to encrypt
out	<i>ciphertext</i>	Output cipher text.
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector
	<i>ivSize</i>	Size of the IV
	<i>aad</i>	Input additional authentication data
	<i>aadSize</i>	Input size in bytes of AAD
	<i>key</i>	Input key to use for encryption

	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
out	<i>tag</i>	Output hash tag. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16.

Returns

Status from job descriptor push

18.12.5.2.9 `status_t CAAM_AES_DecryptTagGcmNonBlocking (CAAM_Type * base,
caam_handle_t * handle, caam_desc_aes_gcm_t descriptor, const uint8_t *
ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv, size_t ivSize, const
uint8_t * aad, size_t aadSize, const uint8_t * key, size_t keySize, const uint8_t * tag,
size_t tagSize)`

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the G-HASH is calculated and compared with the received GHASH in 'tag' field. Puts AES GCM decrypt and check tag descriptor to CAAM input job ring.

Parameters

	<i>base</i>	CAAM peripheral base address
	<i>handle</i>	Handle used for this request. Specifies jobRing.
out	<i>descriptor</i>	Memory for the CAAM descriptor.
	<i>ciphertext</i>	Input cipher text to decrypt
out	<i>plaintext</i>	Output plain text.
	<i>size</i>	Size of input and output data in bytes
	<i>iv</i>	Input initial vector
	<i>ivSize</i>	Size of the IV
	<i>aad</i>	Input additional authentication data
	<i>aadSize</i>	Input size in bytes of AAD
	<i>key</i>	Input key to use for encryption

	<i>keySize</i>	Size of the input key, in bytes. Must be 16, 24, or 32.
	<i>tag</i>	Input hash tag to compare. Set to NULL to skip tag processing.
	<i>tagSize</i>	Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16.

Returns

Status from job descriptor push

Chapter 19

CACHE: ARMV7-M7 CACHE Memory Controller

19.1 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.

19.2 Function groups

19.2.1 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.

19.2.2 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 it's 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 pl310)

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.

19.3 Macro Definition Documentation

19.3.1 #define `FSL_CACHE_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 4)`)

19.4 Function Documentation

19.4.1 void L1CACHE_InvalidateICacheByRange (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_BYTE) 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.

19.4.2 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.

19.4.3 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.

19.4.4 `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.

19.4.5 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.

19.4.6 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.

19.4.7 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.

19.4.8 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 20

CACHE: LMEM CACHE Memory Controller

20.1 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 in this arch is the previous the local memory controller (LMEM).

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 only L1 driver APIs.

20.2 Function groups

20.2.1 L1 CACHE Operation

The L1 CACHE has both code cache and data cache. This function group provides two independent API groups 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.

Macros

- #define [L1CODEBUSCACHE_LINESIZE_BYTE](#) FSL_FEATURE_L1ICACHE_LINESIZE_BYTE
code bus cache line size is equal to system bus line size, so the unified I/D cache line size equals too.
- #define [L1SYSTEMBUSCACHE_LINESIZE_BYTE](#) [L1CODEBUSCACHE_LINESIZE_BYTE](#)
The system bus CACHE line size is 16B = 128b.

Driver version

- #define [FSL_CACHE_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 6))
cache driver version.

cache control for L1 cache (local memory controller for code/system bus cache)

- void [L1CACHE_EnableCodeCache](#) (void)
Enables the processor code bus cache.
- void [L1CACHE_DisableCodeCache](#) (void)
Disables the processor code bus cache.

- void [L1CACHE_InvalidateCodeCache](#) (void)
Invalidates the processor code bus cache.
- void [L1CACHE_InvalidateCodeCacheByRange](#) (uint32_t address, uint32_t size_byte)
Invalidates processor code bus cache by range.
- void [L1CACHE_CleanCodeCache](#) (void)
Cleans the processor code bus cache.
- void [L1CACHE_CleanCodeCacheByRange](#) (uint32_t address, uint32_t size_byte)
Cleans processor code bus cache by range.
- void [L1CACHE_CleanInvalidateCodeCache](#) (void)
Cleans and invalidates the processor code bus cache.
- void [L1CACHE_CleanInvalidateCodeCacheByRange](#) (uint32_t address, uint32_t size_byte)
Cleans and invalidate processor code bus cache by range.
- static void [L1CACHE_EnableCodeCacheWriteBuffer](#) (bool enable)
Enables/disables the processor code bus write buffer.
- void [L1CACHE_EnableSystemCache](#) (void)
Enables the processor system bus cache.
- void [L1CACHE_DisableSystemCache](#) (void)
Disables the processor system bus cache.
- void [L1CACHE_InvalidateSystemCache](#) (void)
Invalidates the processor system bus cache.
- void [L1CACHE_InvalidateSystemCacheByRange](#) (uint32_t address, uint32_t size_byte)
Invalidates processor system bus cache by range.
- void [L1CACHE_CleanSystemCache](#) (void)
Cleans the processor system bus cache.
- void [L1CACHE_CleanSystemCacheByRange](#) (uint32_t address, uint32_t size_byte)
Cleans processor system bus cache by range.
- void [L1CACHE_CleanInvalidateSystemCache](#) (void)
Cleans and invalidates the processor system bus cache.
- void [L1CACHE_CleanInvalidateSystemCacheByRange](#) (uint32_t address, uint32_t size_byte)
Cleans and Invalidates processor system bus cache by range.
- static void [L1CACHE_EnableSystemCacheWriteBuffer](#) (bool enable)
Enables/disables the processor system bus write buffer.

cache control for unified L1 cache driver

- void [L1CACHE_InvalidateICacheByRange](#) (uint32_t address, uint32_t size_byte)
Invalidates cortex-m4 L1 instruction cache by range.
- static void [L1CACHE_InvalidateDCacheByRange](#) (uint32_t address, uint32_t size_byte)
Invalidates cortex-m4 L1 data cache by range.
- void [L1CACHE_CleanDCacheByRange](#) (uint32_t address, uint32_t size_byte)
Cleans cortex-m4 L1 data cache by range.
- void [L1CACHE_CleanInvalidateDCacheByRange](#) (uint32_t address, uint32_t size_byte)
Cleans and Invalidates cortex-m4 L1 data 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.

20.3 Macro Definition Documentation

20.3.1 #define FSL_CACHE_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))

20.3.2 #define L1CODEBUSCACHE_LINESIZE_BYTE FSL_FEATURE_L1ICACHE_LINESIZE_BYTE

The code bus CACHE line size is 16B = 128b.

20.3.3 #define L1SYSTEMBUSCACHE_LINESIZE_BYTE L1CODEBUSCACHE_LINESIZE_BYTE

20.4 Function Documentation

20.4.1 void L1CACHE_InvalidateCodeCacheByRange (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.

Note

Address and size should be aligned to "L1CODCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_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.

20.4.2 void L1CACHE_CleanCodeCacheByRange (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.

Note

Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_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.

20.4.3 void L1CACHE_CleanInvalidateCodeCacheByRange (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.

Note

Address and size should be aligned to "L1CODEBUSCACHE_LINESIZE_BYTE". The startAddr here will be forced to align to L1CODEBUSCACHE_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.

20.4.4 static void L1CACHE_EnableCodeCacheWriteBuffer (bool *enable*) [inline], [static]

Parameters

<i>enable</i>	The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.
---------------	---

20.4.5 void L1CACHE_InvalidateSystemCacheByRange (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.

Note

Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The start-Addr here will be forced to align to L1SYSTEMBUSCACHE_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.

20.4.6 void L1CACHE_CleanSystemCacheByRange (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.

Note

Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The start-Addr here will be forced to align to L1SYSTEMBUSCACHE_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.

20.4.7 void L1CACHE_CleanInvalidateSystemCacheByRange (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 Clean and Invalidated.

Note

Address and size should be aligned to "L1SYSTEMBUSCACHE_LINESIZE_BYTE". The start-Addr here will be forced to align to L1SYSTEMBUSCACHE_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.

20.4.8 `static void L1CACHE_EnableSystemCacheWriteBuffer (bool enable)`
`[inline], [static]`

Parameters

<i>enable</i>	The enable or disable flag. true - enable the code bus write buffer. false - disable the code bus write buffer.
---------------	---

20.4.9 void L1CACHE_InvalidateICacheByRange (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 16-Byte(FSL_FEATURE_L1ICACHE_LINESIZE_BYTE) aligned.

20.4.10 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 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

20.4.11 void L1CACHE_CleanDCacheByRange (uint32_t *address*, uint32_t *size_byte*)

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 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

20.4.12 void L1CACHE_CleanInvalidateDCacheByRange (uint32_t *address*, uint32_t *size_byte*)

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 16-Byte(FSL_FEATURE_L1DCACHE_LINESIZE_BYTE) aligned.

20.4.13 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1ICACHE_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.

20.4.14 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_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.

20.4.15 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_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.

20.4.16 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.

Note

Address and size should be aligned to 16-Byte due to the cache operation unit FSL_FEATURE_L1DCACHE_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 21

Common Driver

21.1 Overview

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

Macros

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

Typedefs

- typedef int32_t `status_t`
Type used for all status and error return values.

Enumerations

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


```
kStatusGroup_POWER_MANAGER = 159 }
```

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.

Driver version

- #define [FSL_COMMON_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 3, 1))
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](#)()

21.2 Macro Definition Documentation

21.2.1 #define FSL_DRIVER_TRANSFER_DOUBLE_WEAK_IRQ 1

21.2.2 #define MAKE_STATUS(group, code) (((group)*100L) + (code)))

21.2.3 #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

21.2.4 #define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))

21.2.5 #define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U

21.2.6 #define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U

21.2.7 #define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U

21.2.8 #define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U

21.2.9 #define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U

21.2.10 #define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U

21.2.11 #define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U

21.2.12 #define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U

21.2.13 #define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U

21.2.14 #define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U

21.2.15 #define DEBUG_CONSOLE_DEVICE_TYPE_QSCI 10U

21.2.16 #define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))

21.3 Typedef Documentation

21.3.1 typedef int32_t status_t

21.4 Enumeration Type Documentation

21.4.1 enum _status_groups

Enumerator

kStatusGroup_Generic Group number for generic status codes.
kStatusGroup_FLASH Group number for FLASH status codes.
kStatusGroup_LPSPI Group number for LPSPI status codes.
kStatusGroup_FLEXIO_SPI Group number for FLEXIO SPI status codes.
kStatusGroup_DSPI Group number for DSPI status codes.
kStatusGroup_FLEXIO_UART Group number for FLEXIO UART status codes.
kStatusGroup_FLEXIO_I2C Group number for FLEXIO I2C status codes.
kStatusGroup_LPI2C Group number for LPI2C status codes.
kStatusGroup_UART Group number for UART status codes.
kStatusGroup_I2C Group number for I2C status codes.
kStatusGroup_LPSCI Group number for LPSCI status codes.
kStatusGroup_LPUART Group number for LPUART status codes.
kStatusGroup_SPI Group number for SPI status code.
kStatusGroup_XRDC Group number for XRDC status code.
kStatusGroup_SEMA42 Group number for SEMA42 status code.
kStatusGroup_SDHC Group number for SDHC status code.
kStatusGroup_SDMMC Group number for SDMMC status code.
kStatusGroup_SAI Group number for SAI status code.
kStatusGroup_MCG Group number for MCG status codes.
kStatusGroup_SCG Group number for SCG status codes.
kStatusGroup_SDSPI Group number for SDSPI status codes.
kStatusGroup_FLEXIO_I2S Group number for FLEXIO I2S status codes.
kStatusGroup_FLEXIO_MCULCD Group number for FLEXIO LCD status codes.
kStatusGroup_FLASHIAP Group number for FLASHIAP status codes.
kStatusGroup_FLEXCOMM_I2C Group number for FLEXCOMM I2C status codes.
kStatusGroup_I2S Group number for I2S status codes.
kStatusGroup_IUART Group number for IUART status codes.
kStatusGroup_CSI Group number for CSI status codes.
kStatusGroup_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_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_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_SNT Group number for SNT status codes.

kStatusGroup_QUEUEDSPI Group number for QSPI status codes.

kStatusGroup_POWER_MANAGER Group number for POWER_MANAGER status codes.

21.4.2 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.

21.5 Function Documentation

21.5.1 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.
------------	-------------------

21.5.2 void SDK_Free (void * *ptr*)

Parameters

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

21.5.3 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.

Chapter 22

CSI: CMOS Sensor Interface

22.1 Overview

The MCUXpresso SDK provides a driver for the CMOS Sensor Interface (CSI)

The CSI enables the chip to connect directly to external CMOS image sensors. The CSI driver provides functional APIs and transactional APIs for the CSI module. The functional APIs implement the basic functions, so the user can construct them for a special use case. The transactional APIs provide a queue mechanism in order for the user to submit an empty frame buffer and get a fully-filled frame buffer easily.

22.2 Frame Buffer Queue

The CSI transactional functions maintain a frame buffer queue. The queue size is defined by the macro `CSI_DRIVER_QUEUE_SIZE`. The queue size is 4 by default, but the user can override it by redefining the macro value in the project setting.

To use transactional APIs, first call [CSI_TransferCreateHandle](#) to create a handle to save the CSI driver state. This function initializes the frame buffer queue to empty status.

After the handle is created, the function [CSI_TransferSubmitEmptyBuffer](#) can be used to submit the empty frame buffer to the queue. If the queue does not have room to save the new empty frame buffers, this function returns with an error. It is not necessary to check the queue rooms before submitting an empty frame buffer. After this step, the application can call [CSI_TransferStart](#) to start the transfer. There must be at least two empty buffers in the queue, otherwise this function returns an error. The incoming frames are saved to the empty buffers one by one, and a callback is provided when every frame completed. To get the fully-filled frame buffer, call the function [CSI_TransferGetFullBuffer](#). This function returns an error if the frame buffer queue does not have full buffers. Therefore, it is not necessary to check the full buffer number in the queue before this function.

To stop the transfer, call the function [CSI_TransferStop](#) at anytime. If the queue has some full frame buffers, the application can still read them out after this stop function.

Once the transfer is started by calling [CSI_TransferStart](#), the CSI device starts to receive frames and save into buffer. The CSI devices does not stop until [CSI_TransferStop](#) is called. If application does not submit empty buffer to CSI driver, the CSI driver always writes to the last submitted empty buffer, this buffer will never be sent into full buffer queue until new empty buffer submitted. In other words, one frame buffer is reserved by CSI driver, if application submits N empty buffers, it could get (N-1) full buffers.

22.3 Fragment Mode

The frame buffer queue mechanism needs large memory, it is not suitable for some special case, for example, no SDRAM used. Fragment mode is designed for this purpose, it needs two types of buffers:

1. DMA buffer. It could be as small as (camera frame width x 2 x 2) bytes, CSI DMA writes the input data to this buffer.

2. Frame buffer. The input data is copied to this buffer at last. What is more, user could define a window (in other words, region of interest), only image in this window will be copied to the frame buffer. If input data is YUV422 format, user can only save Y component optionally.

Limitations:

1. Fragment mode could not be used together with frame buffer queue mode.
2. In fragment mode, user should pay attention to the system payload. When the payload is high, the image capture might be broken.

22.4 Typical use case

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

Data Structures

- struct `csi_config_t`
Configuration to initialize the CSI module. [More...](#)
- struct `csi_handle_t`
CSI handle structure. [More...](#)

Macros

- #define `CSI_DRIVER_QUEUE_SIZE` 4U
Size of the frame buffer queue used in CSI transactional function.
- #define `CSI_DRIVER_FRAG_MODE` 0U
Enable fragment capture function or not.

Typedefs

- typedef void(* `csi_transfer_callback_t`)(CSI_Type *base, csi_handle_t *handle, `status_t` status, void *userData)
CSI transfer callback function.

Enumerations

- enum {
 `kStatus_CSI_NoEmptyBuffer` = MAKE_STATUS(kStatusGroup_CSI, 0),
 `kStatus_CSI_NoFullBuffer` = MAKE_STATUS(kStatusGroup_CSI, 1),
 `kStatus_CSI_QueueFull` = MAKE_STATUS(kStatusGroup_CSI, 2),
 `kStatus_CSI_FrameDone` = MAKE_STATUS(kStatusGroup_CSI, 3) }
Error codes for the CSI driver.
- enum `csi_work_mode_t` {
 `kCSI_GatedClockMode` = CSI_CR1_GCLK_MODE(1U),
 `kCSI_NonGatedClockMode` = 0U,
 `kCSI_CCIR656ProgressiveMode` = CSI_CR1_CCIR_EN(1U) }
CSI work mode.

- enum `csi_data_bus_t` {
`kCSI_DataBus8Bit`,
`kCSI_DataBus16Bit`,
`kCSI_DataBus24Bit` }
CSI data bus width.
- enum `_csi_polarity_flags` {
`kCSI_HsyncActiveLow` = 0U,
`kCSI_HsyncActiveHigh` = CSI_CR1_HSYNC_POL_MASK,
`kCSI_DataLatchOnRisingEdge` = CSI_CR1_REDGE_MASK,
`kCSI_DataLatchOnFallingEdge` = 0U,
`kCSI_VsyncActiveHigh` = 0U,
`kCSI_VsyncActiveLow` = CSI_CR1_SOF_POL_MASK }
CSI signal polarity.
- enum `csi_fifo_t` {
`kCSI_RxFifo` = (1U << 0U),
`kCSI_StatFifo` = (1U << 1U),
`kCSI_AllFifo` = 0x01 | 0x02 }
The CSI FIFO, used for FIFO operation.
- enum `_csi_interrupt_enable` {
`kCSI_EndOfFrameInterruptEnable` = CSI_CR1_EOF_INT_EN_MASK,
`kCSI_ChangeOfFieldInterruptEnable` = CSI_CR1_COF_INT_EN_MASK,
`kCSI_StatFifoOverrunInterruptEnable` = CSI_CR1_SF_OR_INTEN_MASK,
`kCSI_RxFifoOverrunInterruptEnable` = CSI_CR1_RF_OR_INTEN_MASK,
`kCSI_StatFifoDmaDoneInterruptEnable` = CSI_CR1_SFF_DMA_DONE_INTEN_MASK,
`kCSI_StatFifoFullInterruptEnable` = CSI_CR1_STATFF_INTEN_MASK,
`kCSI_RxBuffer1DmaDoneInterruptEnable` = CSI_CR1_FB2_DMA_DONE_INTEN_MASK,
`kCSI_RxBuffer0DmaDoneInterruptEnable` = CSI_CR1_FB1_DMA_DONE_INTEN_MASK,
`kCSI_RxFifoFullInterruptEnable` = CSI_CR1_RXFF_INTEN_MASK,
`kCSI_StartOfFrameInterruptEnable` = CSI_CR1_SOF_INTEN_MASK,
`kCSI_EccErrorInterruptEnable` = CSI_CR3_ECC_INT_EN_MASK,
`kCSI_AhbResErrorInterruptEnable` = CSI_CR3_HRESP_ERR_EN_MASK,
`kCSI_BaseAddrChangeErrorInterruptEnable` = CSI_CR18_BASEADDR_CHANGE_ERROR_IE-
_MASK << 6U,
`kCSI_Field0DoneInterruptEnable` = CSI_CR18_FIELD0_DONE_IE_MASK << 6U,
`kCSI_Field1DoneInterruptEnable` = CSI_CR18_DMA_FIELD1_DONE_IE_MASK << 6U }
CSI feature interrupt source.
- enum `_csi_flags` {

```

kCSI_RxFifoDataReadyFlag = CSI_SR_DRDY_MASK,
kCSI_EccErrorFlag = CSI_SR_ECC_INT_MASK,
kCSI_AhbResErrorFlag = CSI_SR_HRESP_ERR_INT_MASK,
kCSI_ChangeOfFieldFlag = CSI_SR_COF_INT_MASK,
kCSI_Field0PresentFlag = CSI_SR_F1_INT_MASK,
kCSI_Field1PresentFlag = CSI_SR_F2_INT_MASK,
kCSI_StartOfFrameFlag = CSI_SR_SOF_INT_MASK,
kCSI_EndOfFrameFlag = CSI_SR_EOF_INT_MASK,
kCSI_RxFifoFullFlag = CSI_SR_RxFF_INT_MASK,
kCSI_RxBuffer1DmaDoneFlag = CSI_SR_DMA_TSF_DONE_FB2_MASK,
kCSI_RxBuffer0DmaDoneFlag = CSI_SR_DMA_TSF_DONE_FB1_MASK,
kCSI_StatFifoFullFlag = CSI_SR_STATFF_INT_MASK,
kCSI_StatFifoDmaDoneFlag = CSI_SR_DMA_TSF_DONE_SFF_MASK,
kCSI_StatFifoOverrunFlag = CSI_SR_SF_OR_INT_MASK,
kCSI_RxFifoOverrunFlag = CSI_SR_RF_OR_INT_MASK,
kCSI_Field0DoneFlag = CSI_SR_DMA_FIELD0_DONE_MASK,
kCSI_Field1DoneFlag = CSI_SR_DMA_FIELD1_DONE_MASK,
kCSI_BaseAddrChangeErrorFlag = CSI_SR_BASEADDR_CHHANGE_ERROR_MASK }
    CSI status flags.

```

Driver version

- #define **FSL_CSI_DRIVER_VERSION** ([MAKE_VERSION](#)(2, 1, 5))

Initialization and deinitialization

- [status_t](#) **CSI_Init** (CSI_Type *base, const [csi_config_t](#) *config)
Initialize the CSI.
- void **CSI_Deinit** (CSI_Type *base)
De-initialize the CSI.
- void **CSI_Reset** (CSI_Type *base)
Reset the CSI.
- void **CSI_GetDefaultConfig** ([csi_config_t](#) *config)
Get the default configuration for to initialize the CSI.

Module operation

- void **CSI_ClearFifo** (CSI_Type *base, [csi_fifo_t](#) fifo)
Clear the CSI FIFO.
- void **CSI_ReflashFifoDma** (CSI_Type *base, [csi_fifo_t](#) fifo)
Reflash the CSI FIFO DMA.
- void **CSI_EnableFifoDmaRequest** (CSI_Type *base, [csi_fifo_t](#) fifo, bool enable)
Enable or disable the CSI FIFO DMA request.
- static void **CSI_Start** (CSI_Type *base)
Start to receive data.
- static void **CSI_Stop** (CSI_Type *base)
Stop to receiving data.
- void **CSI_SetRxBufferAddr** (CSI_Type *base, uint8_t index, uint32_t addr)
Set the RX frame buffer address.

Interrupts

- void [CSI_EnableInterrupts](#) (CSI_Type *base, uint32_t mask)
Enables CSI interrupt requests.
- void [CSI_DisableInterrupts](#) (CSI_Type *base, uint32_t mask)
Disable CSI interrupt requests.

Status

- static uint32_t [CSI_GetStatusFlags](#) (CSI_Type *base)
Gets the CSI status flags.
- static void [CSI_ClearStatusFlags](#) (CSI_Type *base, uint32_t statusMask)
Clears the CSI status flag.

Transactional

- [status_t CSI_TransferCreateHandle](#) (CSI_Type *base, csi_handle_t *handle, [csi_transfer_callback_t](#) callback, void *userData)
Initializes the CSI handle.
- [status_t CSI_TransferStart](#) (CSI_Type *base, csi_handle_t *handle)
Start the transfer using transactional functions.
- [status_t CSI_TransferStop](#) (CSI_Type *base, csi_handle_t *handle)
Stop the transfer using transactional functions.
- [status_t CSI_TransferSubmitEmptyBuffer](#) (CSI_Type *base, csi_handle_t *handle, uint32_t frame-Buffer)
Submit empty frame buffer to queue.
- [status_t CSI_TransferGetFullBuffer](#) (CSI_Type *base, csi_handle_t *handle, uint32_t *frame-Buffer)
Get one full frame buffer from queue.
- void [CSI_TransferHandleIRQ](#) (CSI_Type *base, csi_handle_t *handle)
CSI IRQ handle function.

22.5 Data Structure Documentation

22.5.1 struct csi_config_t

Data Fields

- uint16_t [width](#)
Pixels of the input frame.
- uint16_t [height](#)
Lines of the input frame.
- uint32_t [polarityFlags](#)
Timing signal polarity flags, OR'ed value of [_csi_polarity_flags](#).
- uint8_t [bytesPerPixel](#)
Bytes per pixel, valid values are:
- uint16_t [linePitch_Bytes](#)
Frame buffer line pitch, must be 8-byte aligned.
- [csi_work_mode_t](#) [workMode](#)
CSI work mode.

- `csi_data_bus_t dataBus`
Data bus width.
- `bool useExtVsync`
In CCIR656 progressive mode, set true to use external VSYNC signal, set false to use internal VSYNC signal decoded from SOF.

Field Documentation

- (1) `uint16_t csi_config_t::width`
- (2) `uint16_t csi_config_t::height`
- (3) `uint32_t csi_config_t::polarityFlags`
- (4) `uint8_t csi_config_t::bytesPerPixel`
 - 2: Used for RGB565, YUV422, and so on.
 - 4: Used for XRGB8888, XYUV444, and so on.
- (5) `uint16_t csi_config_t::linePitch_Bytes`
- (6) `csi_work_mode_t csi_config_t::workMode`
- (7) `csi_data_bus_t csi_config_t::dataBus`
- (8) `bool csi_config_t::useExtVsync`

22.5.2 struct _csi_handle

Please see the user guide for the details of the CSI driver queue mechanism.

Data Fields

- `uint32_t frameBufferQueue` [CSI_DRIVER_ACTUAL_QUEUE_SIZE]
Frame buffer queue.
- `volatile uint8_t queueWriteIdx`
Pointer to save incoming item.
- `volatile uint8_t queueReadIdx`
Pointer to read out the item.
- `void *volatile emptyBuffer`
Pointer to maintain the empty frame buffers.
- `volatile uint8_t emptyBufferCnt`
Empty frame buffers count.
- `volatile uint8_t activeBufferNum`
How many frame buffers are in progress currently.
- `volatile bool transferStarted`
User has called `CSI_TransferStart` to start frame receiving.
- `csi_transfer_callback_t callback`
Callback function.

- void * [userData](#)
CSI callback function parameter.

Field Documentation

- (1) uint32_t csi_handle_t::frameBufferQueue[CSI_DRIVER_ACTUAL_QUEUE_SIZE]
- (2) volatile uint8_t csi_handle_t::queueWriteldx
- (3) volatile uint8_t csi_handle_t::queueReadldx
- (4) void* volatile csi_handle_t::emptyBuffer
- (5) volatile uint8_t csi_handle_t::emptyBufferCnt
- (6) volatile uint8_t csi_handle_t::activeBufferNum
- (7) volatile bool csi_handle_t::transferStarted
- (8) csi_transfer_callback_t csi_handle_t::callback
- (9) void* csi_handle_t::userData

22.6 Macro Definition Documentation

22.6.1 #define CSI_DRIVER_QUEUE_SIZE 4U

22.6.2 #define CSI_DRIVER_FRAG_MODE 0U

22.7 Typedef Documentation

22.7.1 typedef void(* csi_transfer_callback_t)(CSI_Type *base, csi_handle_t *handle, status_t status, void *userData)

When a new frame is received and saved to the frame buffer queue, the callback is called and the pass the status [kStatus_CSI_FrameDone](#) to upper layer.

22.8 Enumeration Type Documentation

22.8.1 anonymous enum

Enumerator

- kStatus_CSI_NoEmptyBuffer* No empty frame buffer in queue to load to CSI.
- kStatus_CSI_NoFullBuffer* No full frame buffer in queue to read out.
- kStatus_CSI_QueueFull* Queue is full, no room to save new empty buffer.
- kStatus_CSI_FrameDone* New frame received and saved to queue.

22.8.2 enum csi_work_mode_t

The CCIR656 interlace mode is not supported currently.

Enumerator

kCSI_GatedClockMode HSYNC, VSYNC, and PIXCLK signals are used.
kCSI_NonGatedClockMode VSYNC, and PIXCLK signals are used.
kCSI_CCIR656ProgressiveMode CCIR656 progressive mode.

22.8.3 enum csi_data_bus_t

Enumerator

kCSI_DataBus8Bit 8-bit data bus.
kCSI_DataBus16Bit 16-bit data bus.
kCSI_DataBus24Bit 24-bit data bus.

22.8.4 enum _csi_polarity_flags

Enumerator

kCSI_HsyncActiveLow HSYNC is active low.
kCSI_HsyncActiveHigh HSYNC is active high.
kCSI_DataLatchOnRisingEdge Pixel data latched at rising edge of pixel clock.
kCSI_DataLatchOnFallingEdge Pixel data latched at falling edge of pixel clock.
kCSI_VsyncActiveHigh VSYNC is active high.
kCSI_VsyncActiveLow VSYNC is active low.

22.8.5 enum csi_fifo_t

Enumerator

kCSI_RxFifo RXFIFO.
kCSI_StatFifo STAT FIFO.
kCSI_AllFifo Both RXFIFO and STAT FIFO.

22.8.6 enum _csi_interrupt_enable

Enumerator

kCSI_EndOfFrameInterruptEnable End of frame interrupt enable.

kCSI_ChangeOfFieldInterruptEnable Change of field interrupt enable.
kCSI_StatFifoOverrunInterruptEnable STAT FIFO overrun interrupt enable.
kCSI_RxFifoOverrunInterruptEnable RXFIFO overrun interrupt enable.
kCSI_StatFifoDmaDoneInterruptEnable STAT FIFO DMA done interrupt enable.
kCSI_StatFifoFullInterruptEnable STAT FIFO full interrupt enable.
kCSI_RxBuffer1DmaDoneInterruptEnable RX frame buffer 1 DMA transfer done.
kCSI_RxBuffer0DmaDoneInterruptEnable RX frame buffer 0 DMA transfer done.
kCSI_RxFifoFullInterruptEnable RXFIFO full interrupt enable.
kCSI_StartOfFrameInterruptEnable Start of frame (SOF) interrupt enable.
kCSI_EccErrorInterruptEnable ECC error detection interrupt enable.
kCSI_AhbResErrorInterruptEnable AHB response Error interrupt enable.
kCSI_BaseAddrChangeErrorInterruptEnable The DMA output buffer base address changes before DMA completed.
kCSI_Field0DoneInterruptEnable Field 0 done interrupt enable.
kCSI_Field1DoneInterruptEnable Field 1 done interrupt enable.

22.8.7 enum _csi_flags

The following status register flags can be cleared:

- ***kCSI_EccErrorFlag***
- ***kCSI_AhbResErrorFlag***
- ***kCSI_ChangeOfFieldFlag***
- ***kCSI_StartOfFrameFlag***
- ***kCSI_EndOfFrameFlag***
- ***kCSI_RxBuffer1DmaDoneFlag***
- ***kCSI_RxBuffer0DmaDoneFlag***
- ***kCSI_StatFifoDmaDoneFlag***
- ***kCSI_StatFifoOverrunFlag***
- ***kCSI_RxFifoOverrunFlag***
- ***kCSI_Field0DoneFlag***
- ***kCSI_Field1DoneFlag***
- ***kCSI_BaseAddrChangeErrorFlag***

Enumerator

kCSI_RxFifoDataReadyFlag RXFIFO data ready.
kCSI_EccErrorFlag ECC error detected.
kCSI_AhbResErrorFlag Hresponse (AHB bus response) Error.
kCSI_ChangeOfFieldFlag Change of field.
kCSI_Field0PresentFlag Field 0 present in CCIR mode.
kCSI_Field1PresentFlag Field 1 present in CCIR mode.
kCSI_StartOfFrameFlag Start of frame (SOF) detected.
kCSI_EndOfFrameFlag End of frame (EOF) detected.

kCSI_RxFifoFullFlag RXFIFO full (Number of data reaches trigger level).
kCSI_RxBuffer1DmaDoneFlag RX frame buffer 1 DMA transfer done.
kCSI_RxBuffer0DmaDoneFlag RX frame buffer 0 DMA transfer done.
kCSI_StatFifoFullFlag STAT FIFO full (Reach trigger level).
kCSI_StatFifoDmaDoneFlag STAT FIFO DMA transfer done.
kCSI_StatFifoOverrunFlag STAT FIFO overrun.
kCSI_RxFifoOverrunFlag RXFIFO overrun.
kCSI_Field0DoneFlag Field 0 transfer done.
kCSI_Field1DoneFlag Field 1 transfer done.
kCSI_BaseAddrChangeErrorFlag The DMA output buffer base address changes before DMA completed.

22.9 Function Documentation

22.9.1 `status_t CSI_Init (CSI_Type * base, const csi_config_t * config)`

This function enables the CSI peripheral clock, and resets the CSI registers.

Parameters

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

Return values

<i>kStatus_Success</i>	Initialize successfully.
<i>kStatus_InvalidArgument</i>	Initialize failed because of invalid argument.

22.9.2 `void CSI_Deinit (CSI_Type * base)`

This function disables the CSI peripheral clock.

Parameters

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

22.9.3 `void CSI_Reset (CSI_Type * base)`

This function resets the CSI peripheral registers to default status.

Parameters

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

22.9.4 void CSI_GetDefaultConfig (csi_config_t * config)

The default configuration value is:

```
config->width = 320U;
config->height = 240U;
config->polarityFlags = kCSI_HsyncActiveHigh |
    kCSI_DataLatchOnRisingEdge;
config->bytesPerPixel = 2U;
config->linePitch_Bytes = 320U * 2U;
config->workMode = kCSI_GatedClockMode;
config->dataBus = kCSI_DataBus8Bit;
config->useExtVsync = true;
```

Parameters

<i>config</i>	Pointer to the CSI configuration.
---------------	-----------------------------------

22.9.5 void CSI_ClearFifo (CSI_Type * base, csi_fifo_t fifo)

This function clears the CSI FIFO.

Parameters

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

22.9.6 void CSI_ReflashFifoDma (CSI_Type * base, csi_fifo_t fifo)

This function reflashes the CSI FIFO DMA.

For RXFIFO, there are two frame buffers. When the CSI module started, it saves the frames to frame buffer 0 then frame buffer 1, the two buffers will be written by turns. After reflash DMA using this function, the CSI is reset to save frame to buffer 0.

Parameters

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

22.9.7 void CSI_EnableFifoDmaRequest (CSI_Type * *base*, csi_fifo_t *fifo*, bool *enable*)

Parameters

<i>base</i>	CSI peripheral base address.
<i>fifo</i>	The FIFO DMA reques to enable or disable.
<i>enable</i>	True to enable, false to disable.

22.9.8 static void CSI_Start (CSI_Type * *base*) [inline], [static]

Parameters

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

22.9.9 static void CSI_Stop (CSI_Type * *base*) [inline], [static]

Parameters

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

22.9.10 void CSI_SetRxBufferAddr (CSI_Type * *base*, uint8_t *index*, uint32_t *addr*)

Parameters

--	--

<i>base</i>	CSI peripheral base address.
<i>index</i>	Buffer index.
<i>addr</i>	Frame buffer address to set.

22.9.11 void CSI_EnableInterrupts (CSI_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	CSI peripheral base address.
<i>mask</i>	The interrupts to enable, pass in as OR'ed value of _csi_interrupt_enable .

22.9.12 void CSI_DisableInterrupts (CSI_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	CSI peripheral base address.
<i>mask</i>	The interrupts to disable, pass in as OR'ed value of _csi_interrupt_enable .

22.9.13 static uint32_t CSI_GetStatusFlags (CSI_Type * *base*) [inline], [static]

Parameters

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

Returns

status flag, it is OR'ed value of [_csi_flags](#).

22.9.14 static void CSI_ClearStatusFlags (CSI_Type * *base*, uint32_t *statusMask*) [inline], [static]

The flags to clear are passed in as OR'ed value of [_csi_flags](#). The following flags are cleared automatically by hardware:

- [kCSI_RxFifoFullFlag](#),

- [kCSI_StatFifoFullFlag](#),
- [kCSI_Field0PresentFlag](#),
- [kCSI_Field1PresentFlag](#),
- [kCSI_RxFifoDataReadyFlag](#),

Parameters

<i>base</i>	CSI peripheral base address.
<i>statusMask</i>	The status flags mask, OR'ed value of _csi_flags .

22.9.15 **status_t CSI_TransferCreateHandle (CSI_Type * *base*, csi_handle_t * *handle*, csi_transfer_callback_t *callback*, void * *userData*)**

This function initializes CSI handle, it should be called before any other CSI transactional functions.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.
<i>callback</i>	Callback function for CSI transfer.
<i>userData</i>	Callback function parameter.

Return values

<i>kStatus_Success</i>	Handle created successfully.
------------------------	------------------------------

22.9.16 **status_t CSI_TransferStart (CSI_Type * *base*, csi_handle_t * *handle*)**

When the empty frame buffers have been submit to CSI driver using function [CSI_TransferSubmitEmpty-Buffer](#), user could call this function to start the transfer. The incoming frame will be saved to the empty frame buffer, and user could be optionally notified through callback function.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.

Return values

<i>kStatus_Success</i>	Started successfully.
<i>kStatus_CSI_NoEmpty-Buffer</i>	Could not start because no empty frame buffer in queue.

22.9.17 **status_t CSI_TransferStop (CSI_Type * *base*, csi_handle_t * *handle*)**

The driver does not clean the full frame buffers in queue. In other words, after calling this function, user still could get the full frame buffers in queue using function [CSI_TransferGetFullBuffer](#).

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.

Return values

<i>kStatus_Success</i>	Stoped successfully.
------------------------	----------------------

22.9.18 **status_t CSI_TransferSubmitEmptyBuffer (CSI_Type * *base*, csi_handle_t * *handle*, uint32_t *frameBuffer*)**

This function could be called before [CSI_TransferStart](#) or after [CSI_TransferStart](#). If there is no room in queue to store the empty frame buffer, this function returns error.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.
<i>frameBuffer</i>	Empty frame buffer to submit.

Return values

<i>kStatus_Success</i>	Started successfully.
------------------------	-----------------------

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

22.9.19 **status_t CSI_TransferGetFullBuffer (CSI_Type * *base*, csi_handle_t * *handle*, uint32_t * *frameBuffer*)**

After the transfer started using function [CSI_TransferStart](#), the incoming frames will be saved to the empty frame buffers in queue. This function gets the full-filled frame buffer from the queue. If there is no full frame buffer in queue, this function returns error.

Parameters

<i>base</i>	CSI peripheral base address.
<i>handle</i>	Pointer to the handle structure.
<i>frameBuffer</i>	Full frame buffer.

Return values

<i>kStatus_Success</i>	Started successfully.
<i>kStatus_CSI_NoFull-Buffer</i>	There is no full frame buffer in queue.

22.9.20 **void CSI_TransferHandleIRQ (CSI_Type * *base*, csi_handle_t * *handle*)**

This function handles the CSI IRQ request to work with CSI driver transactional APIs.

Parameters

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

Chapter 23

DAC12: 12-bit Digital-to-Analog Converter Driver

23.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit Digital-to-Analog Converter (DAC12) module of MCUXpresso SDK devices.

This DAC is the 12-bit resolution digital-to-analog converters with programmable reference generator output. Its output data items are loaded into a FIFO, so that various FIFO mode can be used to output the value for user-defined sequence.

The DAC driver provides a user-friendly interface to operate the DAC peripheral. The user can initialize/deinitialize the DAC driver, set data into FIFO, or enable the interrupt DMA for special events so that the hardware can process the DAC output data automatically. Also, the configuration for software and hardware trigger are also included in the driver.

23.2 Typical use case

23.2.1 A simple use case to output the user-defined DAC12 value.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/dac12`

23.2.2 Working with the trigger

Once more than one data is filled into the FIFO, the output pointer moves into configured mode when a trigger comes. This trigger can be from software or hardware, and moves one item for each trigger. Also, the interrupt/DMA event can be activated when the output pointer hits to the configured position.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/dac12`

Files

- file [fsl_dac12.h](#)

Data Structures

- struct [dac12_hardware_info_t](#)
DAC12 hardware information. [More...](#)
- struct [dac12_config_t](#)
DAC12 module configuration. [More...](#)

Macros

- #define `DAC12_CR_W1C_FLAGS_MASK` (`DAC_CR_OVFF_MASK` | `DAC_CR_UDFF_MASK`)
Define "write 1 to clear" flags.
- #define `DAC12_CR_ALL_FLAGS_MASK` (`DAC12_CR_W1C_FLAGS_MASK` | `DAC_CR_WMF_MASK` | `DAC_CR_NEMPTF_MASK` | `DAC_CR_FULLF_MASK`)
Define all the flag bits in DACx_CR register.

Enumerations

- enum `_dac12_status_flags` {
`kDAC12_OverflowFlag` = `DAC_CR_OVFF_MASK`,
`kDAC12_UnderflowFlag` = `DAC_CR_UDFF_MASK`,
`kDAC12_WatermarkFlag` = `DAC_CR_WMF_MASK`,
`kDAC12_NearlyEmptyFlag` = `DAC_CR_NEMPTF_MASK`,
`kDAC12_FullFlag` = `DAC_CR_FULLF_MASK` }
DAC12 flags.
- enum `_dac12_interrupt_enable` {
`kDAC12_UnderOrOverflowInterruptEnable` = `DAC_CR_UVIE_MASK`,
`kDAC12_WatermarkInterruptEnable` = `DAC_CR_WTMIE_MASK`,
`kDAC12_NearlyEmptyInterruptEnable` = `DAC_CR_EMPTYIE_MASK`,
`kDAC12_FullInterruptEnable` = `DAC_CR_FULLLIE_MASK` }
DAC12 interrupts.
- enum `dac12_fifo_size_info_t` {
`kDAC12_FIFOSize2` = 0U,
`kDAC12_FIFOSize4` = 1U,
`kDAC12_FIFOSize8` = 2U,
`kDAC12_FIFOSize16` = 3U,
`kDAC12_FIFOSize32` = 4U,
`kDAC12_FIFOSize64` = 5U,
`kDAC12_FIFOSize128` = 6U,
`kDAC12_FIFOSize256` = 7U }
DAC12 FIFO size information provided by hardware.
- enum `dac12_fifo_work_mode_t` {
`kDAC12_FIFODisabled` = 0U,
`kDAC12_FIFOWorkAsNormalMode` = 1U,
`kDAC12_FIFOWorkAsSwingMode` = 2U }
DAC12 FIFO work mode.
- enum `dac12_reference_voltage_source_t` {
`kDAC12_ReferenceVoltageSourceAlt1` = 0U,
`kDAC12_ReferenceVoltageSourceAlt2` = 1U }
DAC12 reference voltage source.
- enum `dac12_fifo_trigger_mode_t` {
`kDAC12_FIFOTriggerByHardwareMode` = 0U,
`kDAC12_FIFOTriggerBySoftwareMode` = 1U }
DAC12 FIFO trigger mode.

- enum `dac12_reference_current_source_t` {
`kDAC12_ReferenceCurrentSourceDisabled` = 0U,
`kDAC12_ReferenceCurrentSourceAlt0` = 1U,
`kDAC12_ReferenceCurrentSourceAlt1` = 2U,
`kDAC12_ReferenceCurrentSourceAlt2` = 3U }
DAC internal reference current source.
- enum `dac12_speed_mode_t` {
`kDAC12_SpeedLowMode` = 0U,
`kDAC12_SpeedMiddleMode` = 1U,
`kDAC12_SpeedHighMode` = 2U }
DAC analog buffer speed mode for conversion.

Driver version

- #define `FSL_DAC12_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 0))
DAC12 driver version 2.1.0.

Initialization and de-initialization

- void `DAC12_GetHardwareInfo` (`DAC_Type` *base, `dac12_hardware_info_t` *info)
Get hardware information about this module.
- void `DAC12_Init` (`DAC_Type` *base, const `dac12_config_t` *config)
Initialize the DAC12 module.
- void `DAC12_GetDefaultConfig` (`dac12_config_t` *config)
Initializes the DAC12 user configuration structure.
- void `DAC12_Deinit` (`DAC_Type` *base)
De-initialize the DAC12 module.
- static void `DAC12_Enable` (`DAC_Type` *base, bool enable)
Enable the DAC12's converter or not.
- static void `DAC12_ResetConfig` (`DAC_Type` *base)
Reset all internal logic and registers.
- static void `DAC12_ResetFIFO` (`DAC_Type` *base)
Reset the FIFO pointers.

Status

- static uint32_t `DAC12_GetStatusFlags` (`DAC_Type` *base)
Get status flags.
- static void `DAC12_ClearStatusFlags` (`DAC_Type` *base, uint32_t flags)
Clear status flags.

Interrupts

- static void `DAC12_EnableInterrupts` (`DAC_Type` *base, uint32_t mask)
Enable interrupts.
- static void `DAC12_DisableInterrupts` (`DAC_Type` *base, uint32_t mask)
Disable interrupts.

DMA control

- static void [DAC12_EnableDMA](#) (DAC_Type *base, bool enable)
Enable DMA or not.

Functional feature

- static void [DAC12_SetData](#) (DAC_Type *base, uint32_t value)
Set data into the entry of FIFO buffer.
- static void [DAC12_DoSoftwareTrigger](#) (DAC_Type *base)
Do trigger the FIFO by software.
- static uint32_t [DAC12_GetFIFOReadPointer](#) (DAC_Type *base)
Get the current read pointer of FIFO.
- static uint32_t [DAC12_GetFIFOWritePointer](#) (DAC_Type *base)
Get the current write pointer of FIFO.

23.3 Data Structure Documentation

23.3.1 struct dac12_hardware_info_t

Data Fields

- [dac12_fifo_size_info_t fifoSizeInfo](#)
The number of words in this device's DAC buffer.

Field Documentation

(1) [dac12_fifo_size_info_t](#) [dac12_hardware_info_t::fifoSizeInfo](#)

23.3.2 struct dac12_config_t

Actually, the most fields are for FIFO buffer.

Data Fields

- uint32_t [fifoWatermarkLevel](#)
FIFO's watermark, the max value can be the hardware FIFO size.
- [dac12_fifo_work_mode_t](#) [fifoWorkMode](#)
FIFO's work mode about pointers.
- [dac12_reference_voltage_source_t](#) [referenceVoltageSource](#)
Select the reference voltage source.
- [dac12_reference_current_source_t](#) [referenceCurrentSource](#)
Select the trigger mode for FIFO.
- [dac12_speed_mode_t](#) [speedMode](#)
Select the speed mode for conversion.
- bool [enableAnalogBuffer](#)
Enable analog buffer for high drive.

Field Documentation

- (1) `uint32_t dac12_config_t::fifoWatermarkLevel`
- (2) `dac12_fifo_work_mode_t dac12_config_t::fifoWorkMode`
- (3) `dac12_reference_voltage_source_t dac12_config_t::referenceVoltageSource`
- (4) `dac12_reference_current_source_t dac12_config_t::referenceCurrentSource`

Select the reference current source.

- (5) `dac12_speed_mode_t dac12_config_t::speedMode`
- (6) `bool dac12_config_t::enableAnalogBuffer`

23.4 Macro Definition Documentation

23.4.1 `#define FSL_DAC12_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

23.4.2 `#define DAC12_CR_W1C_FLAGS_MASK (DAC_CR_OVFF_MASK | DAC_CR_UDFF_MASK)`

23.4.3 `#define DAC12_CR_ALL_FLAGS_MASK (DAC12_CR_W1C_FLAGS_MASK | DAC_CR_WMF_MASK | DAC_CR_NEMPTF_MASK | DAC_CR_FULLF_MASK)`

23.5 Enumeration Type Documentation

23.5.1 `enum _dac12_status_flags`

Enumerator

- kDAC12_OverflowFlag*** FIFO overflow status flag, which indicates that more data has been written into FIFO than it can hold.
- kDAC12_UnderflowFlag*** FIFO underflow status flag, which means that there is a new trigger after the FIFO is nearly empty.
- kDAC12_WatermarkFlag*** FIFO watermark status flag, which indicates the remaining FIFO data is less than the watermark setting.
- kDAC12_NearlyEmptyFlag*** FIFO nearly empty flag, which means there is only one data remaining in FIFO.
- kDAC12_FullFlag*** FIFO full status flag, which means that the FIFO read pointer equals the write pointer, as the write pointer increase.

23.5.2 enum _dac12_interrupt_enable

Enumerator

kDAC12_UnderOrOverflowInterruptEnable Underflow and overflow interrupt enable.

kDAC12_WatermarkInterruptEnable Watermark interrupt enable.

kDAC12_NearlyEmptyInterruptEnable Nearly empty interrupt enable.

kDAC12_FullInterruptEnable Full interrupt enable.

23.5.3 enum dac12_fifo_size_info_t

Enumerator

kDAC12_FIFOSize2 FIFO depth is 2.

kDAC12_FIFOSize4 FIFO depth is 4.

kDAC12_FIFOSize8 FIFO depth is 8.

kDAC12_FIFOSize16 FIFO depth is 16.

kDAC12_FIFOSize32 FIFO depth is 32.

kDAC12_FIFOSize64 FIFO depth is 64.

kDAC12_FIFOSize128 FIFO depth is 128.

kDAC12_FIFOSize256 FIFO depth is 256.

23.5.4 enum dac12_fifo_work_mode_t

Enumerator

kDAC12_FIFODisabled FIFO disabled and only one level buffer is enabled. Any data written from this buffer goes to conversion.

kDAC12_FIFOWorkAsNormalMode Data will first read from FIFO to buffer then go to conversion.

kDAC12_FIFOWorkAsSwingMode In Swing mode, the FIFO must be set up to be full. In Swing back mode, a trigger changes the read pointer to make it swing between the FIFO Full and Nearly Empty state. That is, the trigger increases the read pointer till FIFO is nearly empty and decreases the read pointer till the FIFO is full.

23.5.5 enum dac12_reference_voltage_source_t

Enumerator

kDAC12_ReferenceVoltageSourceAlt1 The DAC selects DACREF_1 as the reference voltage.

kDAC12_ReferenceVoltageSourceAlt2 The DAC selects DACREF_2 as the reference voltage.

23.5.6 enum dac12_fifo_trigger_mode_t

Enumerator

kDAC12_FIFOTriggerByHardwareMode Buffer would be triggered by hardware.

kDAC12_FIFOTriggerBySoftwareMode Buffer would be triggered by software.

23.5.7 enum dac12_reference_current_source_t

Analog module needs reference current to keep working . Such reference current can generated by IP itself, or by on-chip PMC's "reference part". If no current reference be selected, analog module can't working normally ,even when other register can still be assigned, DAC would waste current but no function. To make the DAC work, either kDAC12_ReferenceCurrentSourceAltx should be selected.

Enumerator

kDAC12_ReferenceCurrentSourceDisabled None of reference current source is enabled.

kDAC12_ReferenceCurrentSourceAlt0 Use the internal reference current generated by the module itself.

kDAC12_ReferenceCurrentSourceAlt1 Use the ZTC(Zero Temperature Coefficient) reference current generated by on-chip power management module.

kDAC12_ReferenceCurrentSourceAlt2 Use the PTAT(Proportional To Absolution Temperature) reference current generated by power management module.

23.5.8 enum dac12_speed_mode_t

Enumerator

kDAC12_SpeedLowMode Low speed mode.

kDAC12_SpeedMiddleMode Middle speed mode.

kDAC12_SpeedHighMode High speed mode.

23.6 Function Documentation

23.6.1 void DAC12_GetHardwareInfo (DAC_Type * *base*, dac12_hardware_info_t * *info*)

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>info</i>	Pointer to info structure, see to dac12_hardware_info_t .

23.6.2 void DAC12_Init (DAC_Type * *base*, const dac12_config_t * *config*)

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>config</i>	Pointer to configuration structure, see to dac12_config_t .

23.6.3 void DAC12_GetDefaultConfig (dac12_config_t * *config*)

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

```
* config->fifoWatermarkLevel = 0U;
* config->fifoWorkMode = kDAC12_FIFODisabled;
* config->referenceVoltageSource = kDAC12_ReferenceVoltageSourceAlt1;
* config->fifoTriggerMode = kDAC12_FIFOTriggerByHardwareMode;
* config->referenceCurrentSource = kDAC12_ReferenceCurrentSourceAlt0;
* config->speedMode = kDAC12_SpeedLowMode;
* config->speedMode = false;
* config->currentReferenceInternalTrimValue = 0x4;
*
```

Parameters

<i>config</i>	Pointer to the configuration structure. See "dac12_config_t".
---------------	---

23.6.4 void DAC12_Deinit (DAC_Type * *base*)

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

**23.6.5 static void DAC12_Enable (DAC_Type * *base*, bool *enable*) [inline],
[static]**

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>enable</i>	Enable the DAC12's converter or not.

23.6.6 static void DAC12_ResetConfig (DAC_Type * *base*) [inline], [static]

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

23.6.7 static void DAC12_ResetFIFO (DAC_Type * *base*) [inline], [static]

FIFO pointers should only be reset when the DAC12 is disabled. This function can be used to configure both pointers to the same address to reset the FIFO as empty.

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

23.6.8 static uint32_t DAC12_GetStatusFlags (DAC_Type * *base*) [inline], [static]

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

Returns

Mask of current status flags. See to [_dac12_status_flags](#).

23.6.9 static void DAC12_ClearStatusFlags (DAC_Type * *base*, uint32_t *flags*) [inline], [static]

Note: Not all the flags can be cleared by this API. Several flags need special condition to clear them according to target chip's reference manual document.

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>flags</i>	Mask of status flags to be cleared. See to _dac12_status_flags .

23.6.10 static void DAC12_EnableInterrupts (DAC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>mask</i>	Mask value of interrupts to be enabled. See to _dac12_interrupt_enable .

23.6.11 static void DAC12_DisableInterrupts (DAC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>mask</i>	Mask value of interrupts to be disabled. See to _dac12_interrupt_enable .

23.6.12 static void DAC12_EnableDMA (DAC_Type * *base*, bool *enable*) [inline], [static]

When DMA is enabled, the DMA request will be generated by original interrupts. The interrupts will not be presented on this module at the same time.

23.6.13 static void DAC12_SetData (DAC_Type * *base*, uint32_t *value*) [inline], [static]

When the DAC FIFO is disabled, and the one entry buffer is enabled, the DAC converts the data in the buffer to analog output voltage. Any write to the DATA register will replace the data in the buffer and push data to analog conversion without trigger support. When the DAC FIFO is enabled. Writing data would increase the write pointer of FIFO. Also, the data would be restored into the FIFO buffer.

Parameters

<i>base</i>	DAC12 peripheral base address.
<i>value</i>	Setting value into FIFO buffer.

23.6.14 static void DAC12_DoSoftwareTrigger (DAC_Type * *base*) [inline], [static]

When the DAC FIFO is enabled, and software trigger is used. Doing trigger would increase the read pointer, and the data in the entry pointed by read pointer would be converted as new output.

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

23.6.15 static uint32_t DAC12_GetFIFOReadPointer (DAC_Type * *base*) [inline], [static]

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

Returns

Read pointer index of FIFO buffer.

23.6.16 static uint32_t DAC12_GetFIFOWritePointer (DAC_Type * *base*) [inline], [static]

Parameters

<i>base</i>	DAC12 peripheral base address.
-------------	--------------------------------

Returns

Write pointer index of FIFO buffer

Chapter 24

DMAMUX: Direct Memory Access Multiplexer Driver

24.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access Multiplexer (DMAMUX) of MCUXpresso SDK devices.

24.2 Typical use case

24.2.1 DMAMUX Operation

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

Driver version

- #define [FSL_DMAMUX_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 5))
DMAMUX driver version 2.0.5.

DMAMUX Initialization and de-initialization

- void [DMAMUX_Init](#) (DMAMUX_Type *base)
Initializes the DMAMUX peripheral.
- void [DMAMUX_Deinit](#) (DMAMUX_Type *base)
Deinitializes the DMAMUX peripheral.

DMAMUX Channel Operation

- static void [DMAMUX_EnableChannel](#) (DMAMUX_Type *base, uint32_t channel)
Enables the DMAMUX channel.
- static void [DMAMUX_DisableChannel](#) (DMAMUX_Type *base, uint32_t channel)
Disables the DMAMUX channel.
- static void [DMAMUX_SetSource](#) (DMAMUX_Type *base, uint32_t channel, uint32_t source)
Configures the DMAMUX channel source.
- static void [DMAMUX_EnablePeriodTrigger](#) (DMAMUX_Type *base, uint32_t channel)
Enables the DMAMUX period trigger.
- static void [DMAMUX_DisablePeriodTrigger](#) (DMAMUX_Type *base, uint32_t channel)
Disables the DMAMUX period trigger.
- static void [DMAMUX_EnableAlwaysOn](#) (DMAMUX_Type *base, uint32_t channel, bool enable)
Enables the DMA channel to be always ON.

24.3 Macro Definition Documentation

24.3.1 #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

24.4 Function Documentation

24.4.1 void DMAMUX_Init (DMAMUX_Type * *base*)

This function ungates the DMAMUX clock.

Parameters

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

24.4.2 void DMAMUX_Deinit (DMAMUX_Type * *base*)

This function gates the DMAMUX clock.

Parameters

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

24.4.3 static void DMAMUX_EnableChannel (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function enables the DMAMUX channel.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.

24.4.4 static void DMAMUX_DisableChannel (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function disables the DMAMUX channel.

Note

The user must disable the DMAMUX channel before configuring it.

Parameters

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

<i>channel</i>	DMAMUX channel number.
----------------	------------------------

24.4.5 static void DMAMUX_SetSource (DMAMUX_Type * *base*, uint32_t *channel*, uint32_t *source*) [inline], [static]

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.
<i>source</i>	Channel source, which is used to trigger the DMA transfer.

24.4.6 static void DMAMUX_EnablePeriodTrigger (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function enables the DMAMUX period trigger feature.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.

24.4.7 static void DMAMUX_DisablePeriodTrigger (DMAMUX_Type * *base*, uint32_t *channel*) [inline], [static]

This function disables the DMAMUX period trigger.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.

24.4.8 static void DMAMUX_EnableAlwaysOn (DMAMUX_Type * *base*, uint32_t *channel*, bool *enable*) [inline], [static]

This function enables the DMAMUX channel always ON feature.

Parameters

<i>base</i>	DMAMUX peripheral base address.
<i>channel</i>	DMAMUX channel number.
<i>enable</i>	Switcher of the always ON feature. "true" means enabled, "false" means disabled.

Chapter 25

eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

25.1 Overview

The MCUXpresso SDK provides a peripheral driver for the enhanced Direct Memory Access (eDMA) of MCUXpresso SDK devices.

25.2 Typical use case

25.2.1 eDMA Operation

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

Data Structures

- struct [edma_config_t](#)
eDMA global configuration structure. [More...](#)
- struct [edma_transfer_config_t](#)
eDMA transfer configuration [More...](#)
- struct [edma_channel_Preemption_config_t](#)
eDMA channel priority configuration [More...](#)
- struct [edma_minor_offset_config_t](#)
eDMA minor offset configuration [More...](#)
- struct [edma_tcd_t](#)
eDMA TCD. [More...](#)
- struct [edma_handle_t](#)
eDMA transfer handle structure [More...](#)

Macros

- #define [DMA_DCHPRI_INDEX](#)(channel) (((channel) & ~0x03U) | (3U - ((channel)&0x03U)))
Compute the offset unit from DCHPRI3.

Typedefs

- typedef void(* [edma_callback](#))(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcds)
Define callback function for eDMA.

Enumerations

- enum `edma_transfer_size_t` {
`kEDMA_TransferSize1Bytes` = 0x0U,
`kEDMA_TransferSize2Bytes` = 0x1U,
`kEDMA_TransferSize4Bytes` = 0x2U,
`kEDMA_TransferSize8Bytes` = 0x3U,
`kEDMA_TransferSize16Bytes` = 0x4U,
`kEDMA_TransferSize32Bytes` = 0x5U }

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_ES_DBE_MASK,
`kEDMA_SourceBusErrorFlag` = DMA_ES_SBE_MASK,
`kEDMA_ScatterGatherErrorFlag` = DMA_ES_SGE_MASK,
`kEDMA_NbytesErrorFlag` = DMA_ES_NCE_MASK,
`kEDMA_DestinationOffsetErrorFlag` = DMA_ES_DOE_MASK,
`kEDMA_DestinationAddressErrorFlag` = DMA_ES_DAE_MASK,
`kEDMA_SourceOffsetErrorFlag` = DMA_ES_SOE_MASK,
`kEDMA_SourceAddressErrorFlag` = DMA_ES_SAE_MASK,
`kEDMA_ErrorChannelFlag` = DMA_ES_ERRCHN_MASK,
`kEDMA_ChannelPriorityErrorFlag` = DMA_ES_CPE_MASK,
`kEDMA_TransferCanceledFlag` = DMA_ES_ECX_MASK,
`kEDMA_ValidFlag` = (int)DMA_ES_VLD_MASK }

_edma_error_status_flags eDMA channel error status flags.

- enum `edma_interrupt_enable_t` {
`kEDMA_ErrorInterruptEnable` = 0x1U,
`kEDMA_MajorInterruptEnable` = DMA_CSR_INTMAJOR_MASK,
`kEDMA_HalfInterruptEnable` = DMA_CSR_INTHALF_MASK }

eDMA interrupt source

- enum `edma_transfer_type_t` {
`kEDMA_MemoryToMemory` = 0x0U,
`kEDMA_PeripheralToMemory`,
`kEDMA_MemoryToPeripheral`,
`kEDMA_PeripheralToPeripheral` }

eDMA transfer type

- enum {
`kStatus_EDMA_QueueFull` = MAKE_STATUS(kStatusGroup_EDMA, 0),
`kStatus_EDMA_Busy` = MAKE_STATUS(kStatusGroup_EDMA, 1) }

_edma_transfer_status eDMA transfer status

Driver version

- #define `FSL_EDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 4, 3))

eDMA driver version

eDMA initialization and de-initialization

- void [EDMA_Init](#) (DMA_Type *base, const [edma_config_t](#) *config)
Initializes the eDMA peripheral.
- void [EDMA_Deinit](#) (DMA_Type *base)
Deinitializes the eDMA peripheral.
- void [EDMA_InstallTCD](#) (DMA_Type *base, uint32_t channel, [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.
- static void [EDMA_EnableContinuousChannelLinkMode](#) (DMA_Type *base, bool enable)
Enable/Disable continuous channel link mode.
- static void [EDMA_EnableMinorLoopMapping](#) (DMA_Type *base, bool enable)
Enable/Disable minor loop mapping.

eDMA Channel Operation

- void [EDMA_ResetChannel](#) (DMA_Type *base, uint32_t channel)
Sets all TCD registers to default values.
- void [EDMA_SetTransferConfig](#) (DMA_Type *base, uint32_t channel, const [edma_transfer_config_t](#) *config, [edma_tcd_t](#) *nextTcd)
Configures the eDMA transfer attribute.
- void [EDMA_SetMinorOffsetConfig](#) (DMA_Type *base, uint32_t channel, const [edma_minor_offset_config_t](#) *config)
Configures the eDMA minor offset feature.
- void [EDMA_SetChannelPreemptionConfig](#) (DMA_Type *base, uint32_t channel, const [edma_channel_preemption_config_t](#) *config)
Configures the eDMA channel preemption feature.
- void [EDMA_SetChannelLink](#) (DMA_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](#) (DMA_Type *base, uint32_t channel, [edma_bandwidth_t](#) bandWidth)
Sets the bandwidth for the eDMA transfer.
- void [EDMA_SetModulo](#) (DMA_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_EnableAsyncRequest](#) (DMA_Type *base, uint32_t channel, bool enable)
Enables an async request for the eDMA transfer.
- static void [EDMA_EnableAutoStopRequest](#) (DMA_Type *base, uint32_t channel, bool enable)
Enables an auto stop request for the eDMA transfer.
- void [EDMA_EnableChannelInterrupts](#) (DMA_Type *base, uint32_t channel, uint32_t mask)
Enables the interrupt source for the eDMA transfer.
- void [EDMA_DisableChannelInterrupts](#) (DMA_Type *base, uint32_t channel, uint32_t mask)
Disables the interrupt source for the eDMA transfer.
- void [EDMA_SetMajorOffsetConfig](#) (DMA_Type *base, uint32_t channel, int32_t sourceOffset, int32_t destOffset)
Configures the eDMA channel TCD major offset feature.

eDMA TCD Operation

- 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](#) ([DMA_Type](#) *base, uint32_t channel)
Enables the eDMA hardware channel request.
- static void [EDMA_DisableChannelRequest](#) ([DMA_Type](#) *base, uint32_t channel)
Disables the eDMA hardware channel request.
- static void [EDMA_TriggerChannelStart](#) ([DMA_Type](#) *base, uint32_t channel)
Starts the eDMA transfer by using the software trigger.

eDMA Channel Status Operation

- uint32_t [EDMA_GetRemainingMajorLoopCount](#) ([DMA_Type](#) *base, uint32_t channel)
Gets the remaining major loop count from the eDMA current channel TCD.
- static uint32_t [EDMA_GetErrorStatusFlags](#) ([DMA_Type](#) *base)
Gets the eDMA channel error status flags.
- uint32_t [EDMA_GetChannelStatusFlags](#) ([DMA_Type](#) *base, uint32_t channel)
Gets the eDMA channel status flags.
- void [EDMA_ClearChannelStatusFlags](#) ([DMA_Type](#) *base, uint32_t channel, uint32_t mask)
Clears the eDMA channel status flags.

eDMA Transactional Operation

- void [EDMA_CreateHandle](#) ([edma_handle_t](#) *handle, [DMA_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_PrepareTransferConfig](#) ([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_PrepareTransfer](#) ([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.
- [status_t](#) [EDMA_SubmitTransfer](#) ([edma_handle_t](#) *handle, const [edma_transfer_config_t](#) *config)
Submits the eDMA transfer request.
- 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_GetUnusedTCDDNumber](#) ([edma_handle_t](#) *handle)
Get unused TCD slot number.
- static uint32_t [EDMA_GetNextTCDDAddress](#) ([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.

25.3 Data Structure Documentation

25.3.1 struct [edma_config_t](#)

Data Fields

- bool [enableContinuousLinkMode](#)
Enable (true) continuous link mode.
- bool [enableHaltOnError](#)
Enable (true) transfer halt on error.
- bool [enableRoundRobinArbitration](#)
Enable (true) round robin channel arbitration method or fixed priority arbitration is used for channel selection.
- bool [enableDebugMode](#)
Enable(true) eDMA debug mode.

Field Documentation

(1) bool [edma_config_t::enableContinuousLinkMode](#)

Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

(2) 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.

(3) bool edma_config_t::enableDebugMode

When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

25.3.2 struct edma_transfer_config_t

This structure configures the source/destination transfer attribute.

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 applied to the current source address to form the next-state value as each source read is completed.
- int16_t [destOffset](#)
Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
- uint32_t [minorLoopBytes](#)
Bytes to transfer in a minor loop.
- uint32_t [majorLoopCounts](#)
Major loop iteration count.

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::srcOffset**

(6) `int16_t edma_transfer_config_t::destOffset`

(7) `uint32_t edma_transfer_config_t::majorLoopCounts`

25.3.3 `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.

25.3.4 `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`

25.3.5 `struct 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.

Data Fields

- `__IO uint32_t SADDR`
SADDR register, used to save source address.
- `__IO uint16_t SOFF`
SOFF register, save offset bytes every transfer.
- `__IO uint16_t ATTR`

- `__IO uint32_t NBYTES`
ATTR register, source/destination transfer size and modulo.
- `__IO uint32_t SLAST`
Nbytes register, minor loop length in bytes.
- `__IO uint32_t DADDR`
SLAST register.
- `__IO uint16_t DOFF`
DADDR register, used for destination address.
- `__IO uint16_t CITER`
DOFF register, used for destination offset.
- `__IO uint32_t DLAST_SGA`
CITER register, current minor loop numbers, for unfinished minor loop.
- `__IO uint16_t CSR`
DLASTSGA register, next tcd address used in scatter-gather mode.
- `__IO uint16_t BITER`
CSR register, for TCD control status.
- `__IO uint16_t BITER`
BITER register, begin minor loop count.

Field Documentation

(1) `__IO uint16_t edma_tcd_t::CITER`

(2) `__IO uint16_t edma_tcd_t::BITER`

25.3.6 struct edma_handle_t

Data Fields

- `edma_callback callback`
Callback function for major count exhausted.
- `void * userData`
Callback function parameter.
- `DMA_Type * base`
eDMA peripheral base address.
- `edma_tcd_t * tcdPool`
Pointer to memory stored TCDs.
- `uint8_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.
- `uint8_t flags`
The status of the current channel.

Field Documentation

(1) **edma_callback edma_handle_t::callback**

(2) **void* edma_handle_t::userData**

(3) **DMA_Type* edma_handle_t::base**

(4) **edma_tcd_t* edma_handle_t::tcdPool**

(5) **uint8_t edma_handle_t::channel**

(6) **volatile int8_t edma_handle_t::header**

Should point to the next TCD to be loaded into the eDMA engine.

(7) **volatile int8_t edma_handle_t::tail**

Should point to the next TCD to be stored into the memory pool.

(8) **volatile int8_t edma_handle_t::tcdUsed**

Should reflect the number of TCDs can be used/loaded in the memory.

(9) **volatile int8_t edma_handle_t::tcdSize**

(10) **uint8_t edma_handle_t::flags**

25.4 Macro Definition Documentation

25.4.1 **#define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 3))**

Version 2.4.3.

25.5 Typedef Documentation

25.5.1 **typedef void(* edma_callback)(struct _edma_handle *handle, void *userData, bool transferDone, uint32_t tcDs)**

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 EDMA 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.

25.6 Enumeration Type Documentation

25.6.1 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.

25.6.2 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.

25.6.3 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.

25.6.4 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.

25.6.5 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

25.6.6 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_ChannelPriorityErrorFlag Channel priority is not unique.
kEDMA_TransferCanceledFlag Transfer cancelled.
kEDMA_ValidFlag No error occurred, this bit is 0. Otherwise, it is 1.

25.6.7 enum edma_interrupt_enable_t

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.

25.6.8 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.

25.6.9 anonymous enum

Enumerator

kStatus_EDMA_QueueFull TCD queue is full.
kStatus_EDMA_Busy Channel is busy and can't handle the transfer request.

25.7 Function Documentation

25.7.1 void EDMA_Init (DMA_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.

25.7.2 void EDMA_Deinit (DMA_Type * *base*)

This function gates the eDMA clock.

Parameters

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

25.7.3 void EDMA_InstallTCD (DMA_Type * *base*, uint32_t *channel*, 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.

25.7.4 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.
---------------	--

25.7.5 static void EDMA_EnableContinuousChannelLinkMode (DMA_Type * *base*, bool *enable*) [inline], [static]

Note

Do not use continuous link mode with a channel linking to itself if there is only one minor loop iteration per service request, for example, if the channel's NBYTES value is the same as either the source or destination size. The same data transfer profile can be achieved by simply increasing the NBYTES value, which provides more efficient, faster processing.

Parameters

<i>base</i>	EDMA peripheral base address.
<i>enable</i>	true is enable, false is disable.

25.7.6 static void EDMA_EnableMinorLoopMapping (DMA_Type * *base*, bool *enable*) [inline], [static]

The TCDn.word2 is redefined to include individual enable fields, an offset field, and the NBYTES field.

Parameters

<i>base</i>	EDMA peripheral base address.
<i>enable</i>	true is enable, false is disable.

25.7.7 void EDMA_ResetChannel (DMA_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.

25.7.8 void EDMA_SetTransferConfig (DMA_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.

25.7.9 void EDMA_SetMinorOffsetConfig (DMA_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.

25.7.10 void EDMA_SetChannelPreemptionConfig (DMA_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.

25.7.11 void EDMA_SetChannelLink (DMA_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"> • kEDMA_LinkNone • kEDMA_MinorLink • kEDMA_MajorLink

<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.

25.7.12 void EDMA_SetBandWidth (DMA_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"> • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

25.7.13 void EDMA_SetModulo (DMA_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.

25.7.14 static void EDMA_EnableAsyncRequest (DMA_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>	The command to enable (true) or disable (false).

25.7.15 static void EDMA_EnableAutoStopRequest (DMA_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).

25.7.16 void EDMA_EnableChannelInterrupts (DMA_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.

25.7.17 void EDMA_DisableChannelInterrupts (DMA_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 <code>edma_interrupt_enable_t</code> type.

25.7.18 void EDMA_SetMajorOffsetConfig (DMA_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.

25.7.19 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.

25.7.20 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 TC-D registers. The STCD 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.

25.7.21 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.

25.7.22 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"> • kEDMA_LinkNone • kEDMA_MinorLink • kEDMA_MajorLink
<i>linkedChannel</i>	The linked channel number.

25.7.23 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"> • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

25.7.24 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.

25.7.25 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).

25.7.26 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 edma_interrupt_enable_t type.

25.7.27 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 edma_interrupt_enable_t type.

25.7.28 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.

25.7.29 static void EDMA_EnableChannelRequest (DMA_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.

25.7.30 static void EDMA_DisableChannelRequest (DMA_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.

25.7.31 static void EDMA_TriggerChannelStart (DMA_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.

25.7.32 `uint32_t EDMA_GetRemainingMajorLoopCount (DMA_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: $\text{RemainingBytes} = \text{RemainingMajorLoopCount} * \text{NBYTES}(\text{initially configured})$

25.7.33 static uint32_t EDMA_GetErrorStatusFlags (DMA_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.

25.7.34 uint32_t EDMA_GetChannelStatusFlags (DMA_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.

25.7.35 void EDMA_ClearChannelStatusFlags (DMA_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 <code>_edma_channel_status_flags</code> type.

25.7.36 void EDMA_CreateHandle (edma_handle_t * *handle*, DMA_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.

25.7.37 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_Submit-Transfer.

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.

25.7.38 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.

25.7.39 void EDMA_PrepareTransferConfig (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).

25.7.40 void EDMA_PrepareTransfer (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).

25.7.41 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.

25.7.42 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.
---------------	----------------------

25.7.43 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.
---------------	----------------------

25.7.44 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.
---------------	---------------------

25.7.45 `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.

25.7.46 `static uint32_t EDMA_GetNextTCDAddress (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.

25.7.47 `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.
---------------	----------------------

Chapter 26

eLCDIF: Enhanced LCD Interface

26.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Enhanced LCD Interface(eLCDIF)

The Enhanced LCD Interface supports MPU mode, VSYNC mode, RGB mode (or DOTCLK mode), and DVI mode. The current eLCDIF driver only supports RGB mode.

26.2 Typical use case

26.2.1 Frame buffer update

The function [ELCDIF_SetNextBufferAddr](#) sets the next frame to show to eLCDIF, the eLCDIF loads the new frame and sets the interrupt [kELCDIF_CurFrameDone](#). If no new frame is set, the old one is displayed.

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

26.2.2 Alpha surface

The alpha surface can be enabled to add an extra overlay on the normal display buffer. In this example, the alpha surface is enabled, and the alpha value is updated after every frame loaded to eLCDIF.

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

Data Structures

- struct [elcdif_pixel_format_reg_t](#)
The register value when using different pixel format. [More...](#)
- struct [elcdif_rgb_mode_config_t](#)
eLCDIF configure structure for RGB mode (DOTCLK mode). [More...](#)
- struct [elcdif_as_buffer_config_t](#)
eLCDIF alpha surface buffer configuration. [More...](#)
- struct [elcdif_as_blend_config_t](#)
eLCDIF alpha surface blending configuration. [More...](#)

Enumerations

- enum `_elcdif_polarity_flags` {
`kELCDIF_VsyncActiveLow` = 0U,
`kELCDIF_VsyncActiveHigh` = LCDIF_VDCTRL0_VSYNC_POL_MASK,
`kELCDIF_HsyncActiveLow` = 0U,
`kELCDIF_HsyncActiveHigh` = LCDIF_VDCTRL0_HSYNC_POL_MASK,
`kELCDIF_DataEnableActiveLow` = 0U,
`kELCDIF_DataEnableActiveHigh` = LCDIF_VDCTRL0_ENABLE_POL_MASK,
`kELCDIF_DriveDataOnFallingClkEdge` = 0U,
`kELCDIF_DriveDataOnRisingClkEdge` = LCDIF_VDCTRL0_DOTCLK_POL_MASK }
eLCDIF signal polarity flags
- enum `_elcdif_interrupt_enable` {
`kELCDIF_BusMasterErrorInterruptEnable` = LCDIF_CTRL1_BM_ERROR_IRQ_EN_MASK,
`kELCDIF_TxFifoOverflowInterruptEnable` = LCDIF_CTRL1_OVERFLOW_IRQ_EN_MASK,
`kELCDIF_TxFifoUnderflowInterruptEnable` = LCDIF_CTRL1_UNDERFLOW_IRQ_EN_MASK,
`kELCDIF_CurFrameDoneInterruptEnable`,
`kELCDIF_VsyncEdgeInterruptEnable` }
The eLCDIF interrupts to enable.
- enum `_elcdif_interrupt_flags` {
`kELCDIF_BusMasterError` = LCDIF_CTRL1_BM_ERROR_IRQ_MASK,
`kELCDIF_TxFifoOverflow` = LCDIF_CTRL1_OVERFLOW_IRQ_MASK,
`kELCDIF_TxFifoUnderflow` = LCDIF_CTRL1_UNDERFLOW_IRQ_MASK,
`kELCDIF_CurFrameDone`,
`kELCDIF_VsyncEdge` = LCDIF_CTRL1_VSYNC_EDGE_IRQ_MASK }
The eLCDIF interrupt status flags.
- enum `_elcdif_status_flags` {
`kELCDIF_LFifoFull` = LCDIF_STAT_LFIFO_FULL_MASK,
`kELCDIF_LFifoEmpty` = LCDIF_STAT_LFIFO_EMPTY_MASK,
`kELCDIF_TxFifoFull` = LCDIF_STAT_TXFIFO_FULL_MASK,
`kELCDIF_TxFifoEmpty` = LCDIF_STAT_TXFIFO_EMPTY_MASK }
eLCDIF status flags
- enum `elcdif_pixel_format_t` {
`kELCDIF_PixelFormatRAW8` = 0,
`kELCDIF_PixelFormatRGB565` = 1,
`kELCDIF_PixelFormatRGB666` = 2,
`kELCDIF_PixelFormatXRGB8888` = 3,
`kELCDIF_PixelFormatRGB888` = 4 }
The pixel format.
- enum `elcdif_lcd_data_bus_t` {
`kELCDIF_DataBus8Bit` = LCDIF_CTRL_LCD_DATABUS_WIDTH(1),
`kELCDIF_DataBus16Bit` = LCDIF_CTRL_LCD_DATABUS_WIDTH(0),
`kELCDIF_DataBus18Bit` = LCDIF_CTRL_LCD_DATABUS_WIDTH(2),
`kELCDIF_DataBus24Bit` = LCDIF_CTRL_LCD_DATABUS_WIDTH(3) }
The LCD data bus type.
- enum `elcdif_as_pixel_format_t` {

```

kELCDIF_AsPixelFormatARGB8888 = 0x0,
kELCDIF_AsPixelFormatRGB888 = 0x4,
kELCDIF_AsPixelFormatARGB1555 = 0x8,
kELCDIF_AsPixelFormatARGB4444 = 0x9,
kELCDIF_AsPixelFormatRGB555 = 0xC,
kELCDIF_AsPixelFormatRGB444 = 0xD,
kELCDIF_AsPixelFormatRGB565 = 0xE }

```

eLCDIF alpha surface pixel format.

- enum `elcdif_alpha_mode_t` {
`kELCDIF_AlphaEmbedded`,
`kELCDIF_AlphaOverride`,
`kELCDIF_AlphaMultiply`,
`kELCDIF_AlphaRop` }

eLCDIF alpha mode during blending.

- enum `elcdif_rop_mode_t` {
`kELCDIF_RopMaskAs` = 0x0,
`kELCDIF_RopMaskNotAs` = 0x1,
`kELCDIF_RopMaskAsNot` = 0x2,
`kELCDIF_RopMergeAs` = 0x3,
`kELCDIF_RopMergeNotAs` = 0x4,
`kELCDIF_RopMergeAsNot` = 0x5,
`kELCDIF_RopNotCopyAs` = 0x6,
`kELCDIF_RopNot` = 0x7,
`kELCDIF_RopNotMaskAs` = 0x8,
`kELCDIF_RopNotMergeAs` = 0x9,
`kELCDIF_RopXorAs` = 0xA,
`kELCDIF_RopNotXorAs` = 0xB }

eLCDIF ROP mode during blending.

- enum `elcdif_lut_t` {
`kELCDIF_Lut0` = 0,
`kELCDIF_Lut1` }

eLCDIF LUT

Driver version

- #define `FSL_ELCDIF_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 4)`)
eLCDIF driver version

eLCDIF initialization and de-initialization

- void `ELCDIF_RgbModeInit` (`LCDIF_Type *base`, const `elcdif_rgb_mode_config_t *config`)
Initializes the eLCDIF to work in RGB mode (DOTCLK mode).
- void `ELCDIF_RgbModeGetDefaultConfig` (`elcdif_rgb_mode_config_t *config`)
Gets the eLCDIF default configuration structure for RGB (DOTCLK) mode.
- void `ELCDIF_Deinit` (`LCDIF_Type *base`)
Deinitializes the eLCDIF peripheral.

Module operation

- void [ELCDIF_RgbModeSetPixelFormat](#) (LCDIF_Type *base, [elcdif_pixel_format_t](#) pixelFormat)
Set the pixel format in RGB (DOTCLK) mode.
- static void [ELCDIF_RgbModeStart](#) (LCDIF_Type *base)
Start to display in RGB (DOTCLK) mode.
- void [ELCDIF_RgbModeStop](#) (LCDIF_Type *base)
Stop display in RGB (DOTCLK) mode and wait until finished.
- static void [ELCDIF_SetNextBufferAddr](#) (LCDIF_Type *base, uint32_t bufferAddr)
Set the next frame buffer address to display.
- void [ELCDIF_Reset](#) (LCDIF_Type *base)
Reset the eLCDIF peripheral.

Status

- static uint32_t [ELCDIF_GetCrcValue](#) (LCDIF_Type *base)
Get the CRC value of the frame sent out.
- static uint32_t [ELCDIF_GetBusMasterErrorAddr](#) (LCDIF_Type *base)
Get the bus master error virtual address.
- static uint32_t [ELCDIF_GetStatus](#) (LCDIF_Type *base)
Get the eLCDIF status.
- static uint32_t [ELCDIF_GetLfifoCount](#) (LCDIF_Type *base)
Get current count in Latency buffer (LFIFO).

Interrupts

- static void [ELCDIF_EnableInterrupts](#) (LCDIF_Type *base, uint32_t mask)
Enables eLCDIF interrupt requests.
- static void [ELCDIF_DisableInterrupts](#) (LCDIF_Type *base, uint32_t mask)
Disables eLCDIF interrupt requests.
- static uint32_t [ELCDIF_GetInterruptStatus](#) (LCDIF_Type *base)
Get eLCDIF interrupt pending status.
- static void [ELCDIF_ClearInterruptStatus](#) (LCDIF_Type *base, uint32_t mask)
Clear eLCDIF interrupt pending status.

LUT

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determines which memory to use.

- static void [ELCDIF_EnableLut](#) (LCDIF_Type *base, bool enable)
Enable or disable the LUT.
- [status_t](#) [ELCDIF_UpdateLut](#) (LCDIF_Type *base, [elcdif_lut_t](#) lut, uint16_t startIndex, const uint32_t *lutData, uint16_t count)
Load the LUT value.

26.3 Data Structure Documentation

26.3.1 struct elcdif_pixel_format_reg_t

These register bits control the pixel format:

- CTRL[DATA_FORMAT_24_BIT]
- CTRL[DATA_FORMAT_18_BIT]
- CTRL[DATA_FORMAT_16_BIT]
- CTRL[WORD_LENGTH]
- CTRL1[BYTE_PACKING_FORMAT]

Data Fields

- uint32_t [regCtrl](#)
Value of register CTRL.
- uint32_t [regCtrl1](#)
Value of register CTRL1.

Field Documentation

(1) uint32_t elcdif_pixel_format_reg_t::regCtrl

(2) uint32_t elcdif_pixel_format_reg_t::regCtrl1

26.3.2 struct elcdif_rgb_mode_config_t

Data Fields

- uint16_t [panelWidth](#)
Display panel width, pixels per line.
- uint16_t [panelHeight](#)
Display panel height, how many lines per panel.
- uint8_t [hsw](#)
HSYNC pulse width.
- uint8_t [hfp](#)
Horizontal front porch.
- uint8_t [hbp](#)
Horizontal back porch.
- uint8_t [vsw](#)
VSYNC pulse width.
- uint8_t [vfp](#)
Vertical front porch.
- uint8_t [vbp](#)
Vertical back porch.
- uint32_t [polarityFlags](#)
OR'ed value of [_elcdif_polarity_flags](#), used to control the signal polarity.
- uint32_t [bufferAddr](#)
Frame buffer address.
- [elcdif_pixel_format_t](#) [pixelFormat](#)

- *Pixel format.*
[elcdif_lcd_data_bus_t dataBus](#)
LCD data bus.

Field Documentation

- (1) `uint16_t elcdif_rgb_mode_config_t::panelWidth`
- (2) `uint16_t elcdif_rgb_mode_config_t::panelHeight`
- (3) `uint8_t elcdif_rgb_mode_config_t::hsw`
- (4) `uint8_t elcdif_rgb_mode_config_t::hfp`
- (5) `uint8_t elcdif_rgb_mode_config_t::hbp`
- (6) `uint8_t elcdif_rgb_mode_config_t::vsw`
- (7) `uint8_t elcdif_rgb_mode_config_t::vfp`
- (8) `uint8_t elcdif_rgb_mode_config_t::vbp`
- (9) `uint32_t elcdif_rgb_mode_config_t::polarityFlags`
- (10) `uint32_t elcdif_rgb_mode_config_t::bufferAddr`
- (11) `elcdif_pixel_format_t elcdif_rgb_mode_config_t::pixelFormat`
- (12) `elcdif_lcd_data_bus_t elcdif_rgb_mode_config_t::dataBus`

26.3.3 struct `elcdif_as_buffer_config_t`

Data Fields

- `uint32_t bufferAddr`
Buffer address.
- [elcdif_as_pixel_format_t pixelFormat](#)
Pixel format.

Field Documentation

- (1) `uint32_t elcdif_as_buffer_config_t::bufferAddr`
- (2) `elcdif_as_pixel_format_t elcdif_as_buffer_config_t::pixelFormat`

26.3.4 struct elcdif_as_blend_config_t

Data Fields

- uint8_t [alpha](#)
User defined alpha value, only used when [alphaMode](#) is [kELCDIF_AlphaOverride](#) or [kELCDIF_AlphaRop](#).
- bool [invertAlpha](#)
Set true to invert the alpha.
- [elcdif_alpha_mode_t](#) [alphaMode](#)
Alpha mode.
- [elcdif_rop_mode_t](#) [ropMode](#)
ROP mode, only valid when [alphaMode](#) is [kELCDIF_AlphaRop](#).

Field Documentation

- (1) uint8_t elcdif_as_blend_config_t::alpha
- (2) bool elcdif_as_blend_config_t::invertAlpha
- (3) elcdif_alpha_mode_t elcdif_as_blend_config_t::alphaMode
- (4) elcdif_rop_mode_t elcdif_as_blend_config_t::ropMode

26.4 Enumeration Type Documentation

26.4.1 enum _elcdif_polarity_flags

Enumerator

- kELCDIF_VsyncActiveLow*** VSYNC active low.
- kELCDIF_VsyncActiveHigh*** VSYNC active high.
- kELCDIF_HsyncActiveLow*** HSYNC active low.
- kELCDIF_HsyncActiveHigh*** HSYNC active high.
- kELCDIF_DataEnableActiveLow*** Data enable line active low.
- kELCDIF_DataEnableActiveHigh*** Data enable line active high.
- kELCDIF_DriveDataOnFallingClkEdge*** Drive data on falling clock edge, capture data on rising clock edge.
- kELCDIF_DriveDataOnRisingClkEdge*** Drive data on falling clock edge, capture data on rising clock edge.

26.4.2 enum _elcdif_interrupt_enable

Enumerator

- kELCDIF_BusMasterErrorInterruptEnable*** Bus master error interrupt.
- kELCDIF_TxFifoOverflowInterruptEnable*** TXFIFO overflow interrupt.

kELCDIF_TxFifoUnderflowInterruptEnable TXFIFO underflow interrupt.
kELCDIF_CurFrameDoneInterruptEnable Interrupt when hardware enters vertical blanking state.
kELCDIF_VsyncEdgeInterruptEnable Interrupt when hardware encounters VSYNC edge.

26.4.3 enum _elcdif_interrupt_flags

Enumerator

kELCDIF_BusMasterError Bus master error interrupt.
kELCDIF_TxFifoOverflow TXFIFO overflow interrupt.
kELCDIF_TxFifoUnderflow TXFIFO underflow interrupt.
kELCDIF_CurFrameDone Interrupt when hardware enters vertical blanking state.
kELCDIF_VsyncEdge Interrupt when hardware encounters VSYNC edge.

26.4.4 enum _elcdif_status_flags

Enumerator

kELCDIF_LFifoFull LFIFO full.
kELCDIF_LFifoEmpty LFIFO empty.
kELCDIF_TxFifoFull TXFIFO full.
kELCDIF_TxFifoEmpty TXFIFO empty.

26.4.5 enum elcdif_pixel_format_t

This enumerator should be defined together with the array s_pixelFormatReg. To support new pixel format, enhance this enumerator and s_pixelFormatReg.

Enumerator

kELCDIF_PixelFormatRAW8 RAW 8 bit, four data use 32 bits.
kELCDIF_PixelFormatRGB565 RGB565, two pixel use 32 bits.
kELCDIF_PixelFormatRGB666 RGB666 unpacked, one pixel uses 32 bits, high byte unused, upper 2 bits of other bytes unused.
kELCDIF_PixelFormatXRGB8888 XRGB8888 unpacked, one pixel uses 32 bits, high byte unused.
kELCDIF_PixelFormatRGB888 RGB888 packed, one pixel uses 24 bits.

26.4.6 enum elcdif_lcd_data_bus_t

Enumerator

kELCDIF_DataBus8Bit 8-bit data bus.
kELCDIF_DataBus16Bit 16-bit data bus, support RGB565.
kELCDIF_DataBus18Bit 18-bit data bus, support RGB666.
kELCDIF_DataBus24Bit 24-bit data bus, support RGB888.

26.4.7 enum elcdif_as_pixel_format_t

Enumerator

kELCDIF_AsPixelFormatARGB8888 32-bit pixels with alpha.
kELCDIF_AsPixelFormatRGB888 32-bit pixels without alpha (unpacked 24-bit format)
kELCDIF_AsPixelFormatARGB1555 16-bit pixels with alpha.
kELCDIF_AsPixelFormatARGB4444 16-bit pixels with alpha.
kELCDIF_AsPixelFormatRGB555 16-bit pixels without alpha.
kELCDIF_AsPixelFormatRGB444 16-bit pixels without alpha.
kELCDIF_AsPixelFormatRGB565 16-bit pixels without alpha.

26.4.8 enum elcdif_alpha_mode_t

Enumerator

kELCDIF_AlphaEmbedded The alpha surface pixel alpha value will be used for blend.
kELCDIF_AlphaOverride The user defined alpha value will be used for blend directly.
kELCDIF_AlphaMultiply The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set to 200, user defined alpha set to 100, then the result alpha is $200 * 100 / 255$.
kELCDIF_AlphaRop Raster operation.

26.4.9 enum elcdif_rop_mode_t

Explanation:

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

Enumerator

kELCDIF_RopMaskAs AS AND PS.
kELCDIF_RopMaskNotAs nAS AND PS.
kELCDIF_RopMaskAsNot AS AND nPS.
kELCDIF_RopMergeAs AS OR PS.
kELCDIF_RopMergeNotAs nAS OR PS.
kELCDIF_RopMergeAsNot AS OR nPS.
kELCDIF_RopNotCopyAs nAS.
kELCDIF_RopNot nPS.
kELCDIF_RopNotMaskAs AS NAND PS.
kELCDIF_RopNotMergeAs AS NOR PS.
kELCDIF_RopXorAs AS XOR PS.
kELCDIF_RopNotXorAs AS XNOR PS.

26.4.10 enum elcdif_lut_t

The Lookup Table (LUT) is used to expand the 8 bits pixel to 24 bits pixel before output to external displayer.

There are two 256x24 bits LUT memory in LCDIF, the LSB of frame buffer address determines which memory to use.

Enumerator

kELCDIF_Lut0 LUT 0.
kELCDIF_Lut1 LUT 1.

26.5 Function Documentation

26.5.1 void ELCDIF_RgbModelnit (LCDIF_Type * *base*, const elcdif_rgb_mode_config_t * *config*)

This function ungates the eLCDIF clock and configures the eLCDIF peripheral according to the configuration structure.

Parameters

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

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

26.5.2 void ELCDIF_RgbModeGetDefaultConfig (elcdif_rgb_mode_config_t * *config*)

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

```
config->panelWidth = 480U;
config->panelHeight = 272U;
config->hsw = 41;
config->hfp = 4;
config->hbp = 8;
config->vsw = 10;
config->vfp = 4;
config->vbp = 2;
config->polarityFlags = kELCDIF_VsyncActiveLow |
                        kELCDIF_HsyncActiveLow |
                        kELCDIF_DataEnableActiveLow |
                        kELCDIF_DriveDataOnFallingClkEdge;
config->bufferAddr = 0U;
config->pixelFormat = kELCDIF_PixelFormatRGB888;
config->dataBus = kELCDIF_DataBus24Bit;
```

Parameters

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

26.5.3 void ELCDIF_Deinit (LCDIF_Type * *base*)

Parameters

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

26.5.4 void ELCDIF_RgbModeSetPixelFormat (LCDIF_Type * *base*, elcdif_pixel_format_t *pixelFormat*)

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>pixelFormat</i>	The pixel format.

26.5.5 static void ELCDIF_RgbModeStart (LCDIF_Type * *base*) [inline], [static]

Parameters

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

26.5.6 void ELCDIF_RgbModeStop (LCDIF_Type * *base*)

Parameters

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

26.5.7 static void ELCDIF_SetNextBufferAddr (LCDIF_Type * *base*, uint32_t *bufferAddr*) [inline], [static]

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>bufferAddr</i>	The frame buffer address to set.

26.5.8 void ELCDIF_Reset (LCDIF_Type * *base*)

Parameters

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

26.5.9 static uint32_t ELCDIF_GetCrcValue (LCDIF_Type * *base*) [inline], [static]

When a frame is sent complete (the interrupt [kELCDIF_CurFrameDone](#) assert), this function can be used to get the CRC value of the frame sent.

Parameters

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

Returns

The CRC value.

Note

The CRC value is dependent on the LCD_DATABUS_WIDTH.

26.5.10 static uint32_t ELCDIF_GetBusMasterErrorAddr (LCDIF_Type * *base*) [inline], [static]

When bus master error occurs (the interrupt kELCDIF_BusMasterError assert), this function can get the virtual address at which the AXI master received an error response from the slave.

Parameters

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

Returns

The error virtual address.

26.5.11 static uint32_t ELCDIF_GetStatus (LCDIF_Type * *base*) [inline], [static]

The status flags are returned as a mask value, application could check the corresponding bit. Example:

```
uint32_t statusFlags;
statusFlags = ELCDIF_GetStatus(LCDIF);

if (kELCDIF_LFifoFull & statusFlags)
{
}

if (kELCDIF_TxFifoEmpty & statusFlags)
{
}
```

Parameters

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

Returns

The mask value of status flags, it is OR'ed value of [_elcdif_status_flags](#).

26.5.12 static uint32_t ELCDIF_GetLFifoCount (LCDIF_Type * *base*) [inline], [static]

Parameters

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

Returns

The LFIFO current count

26.5.13 static void ELCDIF_EnableInterrupts (LCDIF_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>mask</i>	interrupt source, OR'ed value of _elcdif_interrupt_enable .

26.5.14 static void ELCDIF_DisableInterrupts (LCDIF_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

<i>mask</i>	interrupt source, OR'ed value of _elcdif_interrupt_enable.
-------------	--

26.5.15 static uint32_t ELCDIF_GetInterruptStatus (LCDIF_Type * *base*) [inline], [static]

Parameters

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

Returns

Interrupt pending status, OR'ed value of _elcdif_interrupt_flags.

26.5.16 static void ELCDIF_ClearInterruptStatus (LCDIF_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>mask</i>	of the flags to clear, OR'ed value of _elcdif_interrupt_flags.

26.5.17 static void ELCDIF_EnableLut (LCDIF_Type * *base*, bool *enable*) [inline], [static]

Parameters

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

26.5.18 status_t ELCDIF_UpdateLut (LCDIF_Type * *base*, elcdif_lut_t *lut*, uint16_t *startIndex*, const uint32_t * *lutData*, uint16_t *count*)

This function loads the LUT value to the specific LUT memory, user can specify the start entry index.

Parameters

<i>base</i>	eLCDIF peripheral base address.
<i>lut</i>	Which LUT to load.
<i>startIndex</i>	The start index of the LUT entry to update.
<i>lutData</i>	The LUT data to load.
<i>count</i>	Count of <code>lutData</code> .

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Wrong argument.

Chapter 27

ENC: Quadrature Encoder/Decoder

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Quadrature Encoder/Decoder (ENC) module of MCUXpresso SDK devices.

This section describes the programming interface of the ENC Peripheral driver. The ENC driver configures the ENC module and provides a functional interface for the user to build the ENC application.

27.2 Function groups

27.2.1 Initialization and De-initialization

This function group initializes default configuration structure for the ENC counter and initializes ENC counter with the normal configuration and de-initialize ENC module. Some APIs are also created to control the features.

27.2.2 Status

This function group get/clear the ENC status.

27.2.3 Interrupts

This function group enable/disable the ENC interrupts.

27.2.4 Value Operation

This function group get the counter/hold value of positions.

27.3 Typical use case

27.3.1 Polling Configuration

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

Data Structures

- struct [enc_config_t](#)

- Define user configuration structure for ENC module. [More...](#)
- struct `enc_self_test_config_t`
Define configuration structure for self test module. [More...](#)

Enumerations

- enum `_enc_interrupt_enable` {
`kENC_HOMETransitionInterruptEnable` = (1U << 0U),
`kENC_INDEXPulseInterruptEnable` = (1U << 1U),
`kENC_WatchdogTimeoutInterruptEnable` = (1U << 2U),
`kENC_PositionCompareInterruptEnable` = (1U << 3U),
`kENC_SimultBothPhaseChangeInterruptEnable`,
`kENC_PositionRollOverInterruptEnable` = (1U << 5U),
`kENC_PositionRollUnderInterruptEnable` = (1U << 6U) }
Interrupt enable/disable mask.
- enum `_enc_status_flags` {
`kENC_HOMETransitionFlag` = (1U << 0U),
`kENC_INDEXPulseFlag` = (1U << 1U),
`kENC_WatchdogTimeoutFlag` = (1U << 2U),
`kENC_PositionCompareFlag` = (1U << 3U),
`kENC_SimultBothPhaseChangeFlag` = (1U << 4U),
`kENC_PositionRollOverFlag` = (1U << 5U),
`kENC_PositionRollUnderFlag` = (1U << 6U),
`kENC_LastCountDirectionFlag` = (1U << 7U) }
Status flag mask.
- enum `_enc_signal_status_flags` {
`kENC_RawHOMESTatusFlag` = ENC_IMR_HOME_MASK,
`kENC_RawINDEXStatusFlag` = ENC_IMR_INDEX_MASK,
`kENC_RawPHBStatusFlag` = ENC_IMR_PHB_MASK,
`kENC_RawPHAEXStatusFlag` = ENC_IMR_PHA_MASK,
`kENC_FilteredHOMESTatusFlag` = ENC_IMR_FHOM_MASK,
`kENC_FilteredINDEXStatusFlag` = ENC_IMR_FIND_MASK,
`kENC_FilteredPHBStatusFlag` = ENC_IMR_FPHB_MASK,
`kENC_FilteredPHAStatusFlag` = ENC_IMR_FPHA_MASK }
Signal status flag mask.
- enum `enc_home_trigger_mode_t` {
`kENC_HOMETriggerDisabled` = 0U,
`kENC_HOMETriggerOnRisingEdge`,
`kENC_HOMETriggerOnFallingEdge` }
Define HOME signal's trigger mode.
- enum `enc_index_trigger_mode_t` {
`kENC_INDEXTriggerDisabled` = 0U,
`kENC_INDEXTriggerOnRisingEdge`,
`kENC_INDEXTriggerOnFallingEdge` }
Define INDEX signal's trigger mode.
- enum `enc_decoder_work_mode_t` {
`kENC_DecoderWorkAsNormalMode` = 0U,

- `kENC_DecoderWorkAsSignalPhaseCountMode` }
Define type for decoder work mode.
- enum `enc_position_match_mode_t` {
`kENC_POSMATCHOnPositionCounterEqualToComapreValue` = 0U,
`kENC_POSMATCHOnReadingAnyPositionCounter` }
Define type for the condition of POSMATCH pulses.
- enum `enc_revolution_count_condition_t` {
`kENC_RevolutionCountOnINDEXPulse` = 0U,
`kENC_RevolutionCountOnRollOverModulus` }
Define type for determining how the revolution counter (REV) is incremented/decremented.
- enum `enc_self_test_direction_t` {
`kENC_SelfTestDirectionPositive` = 0U,
`kENC_SelfTestDirectionNegative` }
Define type for direction of self test generated signal.
- enum `enc_prescaler_t`
Define prescaler value for clock in CTRL3.

Initialization and De-initialization

- void `ENC_Init` (ENC_Type *base, const `enc_config_t` *config)
Initialization for the ENC module.
- void `ENC_Deinit` (ENC_Type *base)
De-initialization for the ENC module.
- void `ENC_GetDefaultConfig` (`enc_config_t` *config)
Get an available pre-defined settings for ENC's configuration.
- void `ENC_DoSoftwareLoadInitialPositionValue` (ENC_Type *base)
Load the initial position value to position counter.
- void `ENC_SetSelfTestConfig` (ENC_Type *base, const `enc_self_test_config_t` *config)
Enable and configure the self test function.
- void `ENC_EnableWatchdog` (ENC_Type *base, bool enable)
Enable watchdog for ENC module.
- void `ENC_SetInitialPositionValue` (ENC_Type *base, uint32_t value)
Set initial position value for ENC module.

Status

- uint32_t `ENC_GetStatusFlags` (ENC_Type *base)
Get the status flags.
- void `ENC_ClearStatusFlags` (ENC_Type *base, uint32_t mask)
Clear the status flags.
- static uint16_t `ENC_GetSignalStatusFlags` (ENC_Type *base)
Get the signals' real-time status.

Interrupts

- void `ENC_EnableInterrupts` (ENC_Type *base, uint32_t mask)
Enable the interrupts.
- void `ENC_DisableInterrupts` (ENC_Type *base, uint32_t mask)
Disable the interrupts.
- uint32_t `ENC_GetEnabledInterrupts` (ENC_Type *base)

Get the enabled interrupts' flags.

Value Operation

- uint32_t [ENC_GetPositionValue](#) (ENC_Type *base)
Get the current position counter's value.
- uint32_t [ENC_GetHoldPositionValue](#) (ENC_Type *base)
Get the hold position counter's value.
- static uint16_t [ENC_GetPositionDifferenceValue](#) (ENC_Type *base)
Get the position difference counter's value.
- static uint16_t [ENC_GetHoldPositionDifferenceValue](#) (ENC_Type *base)
Get the hold position difference counter's value.
- static uint16_t [ENC_GetRevolutionValue](#) (ENC_Type *base)
Get the position revolution counter's value.
- static uint16_t [ENC_GetHoldRevolutionValue](#) (ENC_Type *base)
Get the hold position revolution counter's value.
- static uint16_t [ENC_GetLastEdgeTimeValue](#) (ENC_Type *base)
Get the last edge time value.
- static uint16_t [ENC_GetHoldLastEdgeTimeValue](#) (ENC_Type *base)
Get the last edge time hold value.
- static uint16_t [ENC_GetPositionDifferencePeriodValue](#) (ENC_Type *base)
Get the position difference period value.
- static uint16_t [ENC_GetPositionDifferencePeriodBufferValue](#) (ENC_Type *base)
Get the position difference period buffer value.
- static uint16_t [ENC_GetHoldPositionDifferencePeriodValue](#) (ENC_Type *base)
Get the position difference period hold value.

27.4 Data Structure Documentation

27.4.1 struct enc_config_t

Data Fields

- bool [enableReverseDirection](#)
Enable reverse direction counting.
- [enc_decoder_work_mode_t](#) [decoderWorkMode](#)
Enable signal phase count mode.
- [enc_home_trigger_mode_t](#) [HOMETriggerMode](#)
Enable HOME to initialize position counters.
- [enc_index_trigger_mode_t](#) [INDEXTriggerMode](#)
Enable INDEX to initialize position counters.
- bool [enableTRIGGERClearPositionCounter](#)
Clear POSD, REV, UPOS and LPOS on rising edge of TRIGGER, or not.
- bool [enableTRIGGERClearHoldPositionCounter](#)
Enable update of hold registers on rising edge of TRIGGER, or not.
- bool [enableWatchdog](#)
Enable the watchdog to detect if the target is moving or not.
- uint16_t [watchdogTimeoutValue](#)
Watchdog timeout count value.
- uint16_t [filterCount](#)

- *Input Filter Sample Count.*
uint16_t [filterSamplePeriod](#)
- *Input Filter Sample Period.*
enc_position_match_mode_t [positionMatchMode](#)
- *The condition of POSMATCH pulses.*
uint32_t [positionCompareValue](#)
- *Position compare value.*
enc_revolution_count_condition_t [revolutionCountCondition](#)
- *Revolution Counter Modulus Enable.*
bool [enableModuloCountMode](#)
- *Enable Modulo Counting.*
uint32_t [positionModulusValue](#)
- *Position modulus value.*
uint32_t [positionInitialValue](#)
- *Position initial value.*
bool [enablePeriodMeasurementFunction](#)
- *Enable period measurement function.*
enc_prescaler_t [prescalerValue](#)
- *The value of prescaler.*

Field Documentation

- (1) **bool enc_config_t::enableReverseDirection**
- (2) **enc_decoder_work_mode_t enc_config_t::decoderWorkMode**
- (3) **enc_home_trigger_mode_t enc_config_t::HOMETriggerMode**
- (4) **enc_index_trigger_mode_t enc_config_t::INDEXTriggerMode**
- (5) **bool enc_config_t::enableTRIGGERClearPositionCounter**
- (6) **bool enc_config_t::enableWatchdog**
- (7) **uint16_t enc_config_t::watchdogTimeoutValue**

It stores the timeout count for the quadrature decoder module watchdog timer. This field is only available when "enableWatchdog" = true. The available value is a 16-bit unsigned number.

- (8) **uint16_t enc_config_t::filterCount**

This value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The value represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. The Available range is 0 - 7.

- (9) **uint16_t enc_config_t::filterSamplePeriod**

This value should be set such that the sampling period is larger than the period of the expected noise. This value represents the sampling period (in IPBus clock cycles) of the decoder input signals. The available

range is 0 - 255.

(10) **enc_position_match_mode_t enc_config_t::positionMatchMode**

(11) **uint32_t enc_config_t::positionCompareValue**

The available value is a 32-bit number.

(12) **enc_revolution_count_condition_t enc_config_t::revolutionCountCondition**

(13) **bool enc_config_t::enableModuloCountMode**

(14) **uint32_t enc_config_t::positionModulusValue**

This value would be available only when "enableModuloCountMode" = true. The available value is a 32-bit number.

(15) **uint32_t enc_config_t::positionInitialValue**

The available value is a 32-bit number.

(16) **bool enc_config_t::enablePeriodMeasurementFunction**

(17) **enc_prescaler_t enc_config_t::prescalerValue**

27.4.2 struct enc_self_test_config_t

The self test module provides a quadrature test signal to the inputs of the quadrature decoder module. This is a factory test feature. It is also useful to customers' software development and testing.

Data Fields

- [enc_self_test_direction_t signalDirection](#)
Direction of self test generated signal.
- [uint16_t signalCount](#)
Hold the number of quadrature advances to generate.
- [uint16_t signalPeriod](#)
Hold the period of quadrature phase in IPBus clock cycles.

Field Documentation

(1) **enc_self_test_direction_t enc_self_test_config_t::signalDirection**

(2) **uint16_t enc_self_test_config_t::signalCount**

The available range is 0 - 255.

(3) uint16_t enc_self_test_config_t::signalPeriod

The available range is 0 - 31.

27.5 Enumeration Type Documentation**27.5.1 enum _enc_interrupt_enable**

Enumerator

kENC_HOMETransitionInterruptEnable HOME interrupt enable.
kENC_INDEXPulseInterruptEnable INDEX pulse interrupt enable.
kENC_WatchdogTimeoutInterruptEnable Watchdog timeout interrupt enable.
kENC_PositionCompareInerruptEnable Position compare interrupt enable.
kENC_SimultBothPhaseChangeInterruptEnable Simultaneous PHASEA and PHASEB change interrupt enable.
kENC_PositionRollOverInterruptEnable Roll-over interrupt enable.
kENC_PositionRollUnderInterruptEnable Roll-under interrupt enable.

27.5.2 enum _enc_status_flags

These flags indicate the counter's events.

Enumerator

kENC_HOMETransitionFlag HOME signal transition interrupt request.
kENC_INDEXPulseFlag INDEX Pulse Interrupt Request.
kENC_WatchdogTimeoutFlag Watchdog timeout interrupt request.
kENC_PositionCompareFlag Position compare interrupt request.
kENC_SimultBothPhaseChangeFlag Simultaneous PHASEA and PHASEB change interrupt request.
kENC_PositionRollOverFlag Roll-over interrupt request.
kENC_PositionRollUnderFlag Roll-under interrupt request.
kENC_LastCountDirectionFlag Last count was in the up direction, or the down direction.

27.5.3 enum _enc_signal_status_flags

These flags indicate the counter's signal.

Enumerator

kENC_RawHOMEStatusFlag Raw HOME input.
kENC_RawINDEXStatusFlag Raw INDEX input.
kENC_RawPHBStatusFlag Raw PHASEB input.

kENC_RawPHAEXStatusFlag Raw PHASEA input.
kENC_FilteredHOMESTatusFlag The filtered version of HOME input.
kENC_FilteredINDEXStatusFlag The filtered version of INDEX input.
kENC_FilteredPHBStatusFlag The filtered version of PHASEB input.
kENC_FilteredPHAStatusFlag The filtered version of PHASEA input.

27.5.4 enum enc_home_trigger_mode_t

The ENC would count the trigger from HOME signal line.

Enumerator

kENC_HOMETriggerDisabled HOME signal's trigger is disabled.
kENC_HOMETriggerOnRisingEdge Use positive going edge-to-trigger initialization of position counters.
kENC_HOMETriggerOnFallingEdge Use negative going edge-to-trigger initialization of position counters.

27.5.5 enum enc_index_trigger_mode_t

The ENC would count the trigger from INDEX signal line.

Enumerator

kENC_INDEXTriggerDisabled INDEX signal's trigger is disabled.
kENC_INDEXTriggerOnRisingEdge Use positive going edge-to-trigger initialization of position counters.
kENC_INDEXTriggerOnFallingEdge Use negative going edge-to-trigger initialization of position counters.

27.5.6 enum enc_decoder_work_mode_t

The normal work mode uses the standard quadrature decoder with PHASEA and PHASEB. When in signal phase count mode, a positive transition of the PHASEA input generates a count signal while the PHASEB input and the reverse direction control the counter direction. If the reverse direction is not enabled, PHASEB = 0 means counting up and PHASEB = 1 means counting down. Otherwise, the direction is reversed.

Enumerator

kENC_DecoderWorkAsNormalMode Use standard quadrature decoder with PHASEA and PHASEB.
kENC_DecoderWorkAsSignalPhaseCountMode PHASEA input generates a count signal while PHASEB input control the direction.

27.5.7 enum enc_position_match_mode_t

Enumerator

kENC_POSMATCHOnPositionCounterEqualToCompareValue POSMATCH pulses when a match occurs between the position counters (POS) and the compare value (COMP).

kENC_POSMATCHOnReadingAnyPositionCounter POSMATCH pulses when any position counter register is read.

27.5.8 enum enc_revolution_count_condition_t

Enumerator

kENC_RevolutionCountOnINDEXPulse Use INDEX pulse to increment/decrement revolution counter.

kENC_RevolutionCountOnRollOverModulus Use modulus counting roll-over/under to increment/decrement revolution counter.

27.5.9 enum enc_self_test_direction_t

Enumerator

kENC_SelfTestDirectionPositive Self test generates the signal in positive direction.

kENC_SelfTestDirectionNegative Self test generates the signal in negative direction.

27.5.10 enum enc_prescaler_t

The clock is prescaled by a value of 2^{PRSC} which means that the prescaler logic can divide the clock by a minimum of 1 and a maximum of 32,768.

27.6 Function Documentation

27.6.1 void ENC_Init (ENC_Type * base, const enc_config_t * config)

This function is to make the initialization for the ENC module. It should be called firstly before any operation to the ENC with the operations like:

- Enable the clock for ENC module.
- Configure the ENC's working attributes.

Parameters

<i>base</i>	ENC peripheral base address.
<i>config</i>	Pointer to configuration structure. See to "enc_config_t".

27.6.2 void ENC_Deinit (ENC_Type * *base*)

This function is to make the de-initialization for the ENC module. It could be called when ENC is no longer used with the operations like:

- Disable the clock for ENC module.

Parameters

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

27.6.3 void ENC_GetDefaultConfig (enc_config_t * *config*)

This function initializes the ENC configuration structure with an available settings, the default value are:

```

*  config->enableReverseDirection      = false;
*  config->decoderWorkMode              = kENC_DecoderWorkAsNormalMode
*  ;
*  config->HOMETriggerMode              = kENC_HOMETriggerDisabled;
*  config->INDEXTriggerMode             = kENC_INDEXTriggerDisabled;
*  config->enableTRIGGERClearPositionCounter = false;
*  config->enableTRIGGERClearHoldPositionCounter = false;
*  config->enableWatchdog                = false;
*  config->watchdogTimeoutValue          = 0U;
*  config->filterCount                   = 0U;
*  config->filterSamplePeriod            = 0U;
*  config->positionMatchMode             =
*  kENC_POSMATCHOnPositionCounterEqualToComapreValue;
*  config->positionCompareValue          = 0xFFFFFFFFU;
*  config->revolutionCountCondition      =
*  kENC_RevolutionCountOnINDEXPulse;
*  config->enableModuloCountMode         = false;
*  config->positionModulusValue          = 0U;
*  config->positionInitialValue          = 0U;
*  config->prescalerValue                 = kENC_ClockDiv1;
*  config->enablePeriodMeasurementFunction = true;
*

```

Parameters

<i>config</i>	Pointer to a variable of configuration structure. See to "enc_config_t".
---------------	--

27.6.4 void ENC_DoSoftwareLoadInitialPositionValue (ENC_Type * *base*)

This function is to transfer the initial position value (UNIT and LINIT) contents to position counter (UP-OS and LPOS), so that to provide the consistent operation the position counter registers.

Parameters

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

27.6.5 void ENC_SetSelfTestConfig (ENC_Type * *base*, const enc_self_test_config_t * *config*)

This function is to enable and configuration the self test function. It controls and sets the frequency of a quadrature signal generator. It provides a quadrature test signal to the inputs of the quadrature decoder module. It is a factory test feature; however, it may be useful to customers' software development and testing.

Parameters

<i>base</i>	ENC peripheral base address.
<i>config</i>	Pointer to configuration structure. See to "enc_self_test_config_t". Pass "NULL" to disable.

27.6.6 void ENC_EnableWatchdog (ENC_Type * *base*, bool *enable*)

Parameters

<i>base</i>	ENC peripheral base address
<i>enable</i>	Enables or disables the watchdog

27.6.7 void ENC_SetInitialPositionValue (ENC_Type * *base*, uint32_t *value*)

Parameters

<i>base</i>	ENC peripheral base address
<i>value</i>	Positive initial value

27.6.8 uint32_t ENC_GetStatusFlags (ENC_Type * *base*)

Parameters

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

Returns

Mask value of status flags. For available mask, see to "_enc_status_flags".

27.6.9 void ENC_ClearStatusFlags (ENC_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ENC peripheral base address.
<i>mask</i>	Mask value of status flags to be cleared. For available mask, see to "_enc_status_flags".

**27.6.10 static uint16_t ENC_GetSignalStatusFlags (ENC_Type * *base*)
[inline], [static]**

Parameters

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

Returns

Mask value of signals' real-time status. For available mask, see to "_enc_signal_status_flags"

27.6.11 void ENC_EnableInterrupts (ENC_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ENC peripheral base address.
<i>mask</i>	Mask value of interrupts to be enabled. For available mask, see to "_enc_interrupt_enable".

27.6.12 void ENC_DisableInterrupts (ENC_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	ENC peripheral base address.
<i>mask</i>	Mask value of interrupts to be disabled. For available mask, see to "_enc_interrupt_enable".

27.6.13 uint32_t ENC_GetEnabledInterrupts (ENC_Type * *base*)

Parameters

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

Returns

Mask value of enabled interrupts.

27.6.14 uint32_t ENC_GetPositionValue (ENC_Type * *base*)

Parameters

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

Returns

Current position counter's value.

27.6.15 uint32_t ENC_GetHoldPositionValue (ENC_Type * *base*)

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

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

Returns

Hold position counter's value.

**27.6.16 static uint16_t ENC_GetPositionDifferenceValue (ENC_Type * *base*)
[inline], [static]**

Parameters

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

Returns

The position difference counter's value.

**27.6.17 static uint16_t ENC_GetHoldPositionDifferenceValue (ENC_Type * *base*)
[inline], [static]**

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

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

Returns

Hold position difference counter's value.

**27.6.18 static uint16_t ENC_GetRevolutionValue (ENC_Type * *base*) [inline],
[static]**

Parameters

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

Returns

The position revolution counter's value.

27.6.19 static uint16_t ENC_GetHoldRevolutionValue (ENC_Type * *base*) [inline], [static]

When any of the counter registers is read, the contents of each counter register is written to the corresponding hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

Parameters

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

Returns

Hold position revolution counter's value.

27.6.20 static uint16_t ENC_GetLastEdgeTimeValue (ENC_Type * *base*) [inline], [static]

Parameters

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

Returns

The last edge time hold value.

27.6.21 static uint16_t ENC_GetHoldLastEdgeTimeValue (ENC_Type * *base*) [inline], [static]

Parameters

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

Returns

The last edge time hold value.

27.6.22 static uint16_t ENC_GetPositionDifferencePeriodValue (ENC_Type * *base*) [inline], [static]

Parameters

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

Returns

The position difference period hold value.

27.6.23 static uint16_t ENC_GetPositionDifferencePeriodBufferValue (ENC_Type * *base*) [inline], [static]

Parameters

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

Returns

The position difference period hold value.

27.6.24 static uint16_t ENC_GetHoldPositionDifferencePeriodValue (ENC_Type * *base*) [inline], [static]

Parameters

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

Returns

The position difference period hold value.

Chapter 28

ENET: Ethernet MAC Driver

28.1 Overview

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

ENET: Ethernet MAC Driver {EthernetMACDriver}

28.2 Operations of Ethernet MAC Driver

28.2.1 MII interface Operation

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

28.2.2 MAC address filter

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

28.2.3 Other Basic control Operations

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

28.2.4 Transactional Operation

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

[ENET_GetRxErrBeforeReadFrame\(\)](#) function after [ENET_GetRxFrameSize\(\)](#) and before [ENET_ReadFrame\(\)](#) functions to get the detailed error information.

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

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

28.2.5 PTP IEEE 1588 Feature Operation

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

The [ENET_Ptp1588Configure\(\)](#) function needs to be called when the `ENET_ENHANCEDBUFFERDESCRIPTOR_MODE` is defined and the IEEE 1588 feature is required.

28.3 Typical use case

28.3.1 ENET Initialization, receive, and transmit operations

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

Modules

- [ENET CMSIS Driver](#)

Data Structures

- struct [enet_rx_bd_struct_t](#)
Defines the receive buffer descriptor structure for the little endian system. [More...](#)
- struct [enet_tx_bd_struct_t](#)
Defines the enhanced transmit buffer descriptor structure for the little endian system. [More...](#)
- struct [enet_data_error_stats_t](#)
Defines the ENET data error statistics structure. [More...](#)
- struct [enet_rx_frame_error_t](#)

- Defines the Rx frame error structure. [More...](#)
- struct [enet_transfer_stats_t](#)
Defines the ENET transfer statistics structure. [More...](#)
- struct [enet_frame_info_t](#)
Defines the frame info structure. [More...](#)
- struct [enet_tx_dirty_ring_t](#)
Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)
- struct [enet_buffer_config_t](#)
Defines the receive buffer descriptor configuration structure. [More...](#)
- struct [enet_intcoalesce_config_t](#)
Defines the interrupt coalescing configure structure. [More...](#)
- struct [enet_avb_config_t](#)
Defines the ENET AVB Configure structure. [More...](#)
- struct [enet_config_t](#)
Defines the basic configuration structure for the ENET device. [More...](#)
- struct [enet_tx_bd_ring_t](#)
Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)
- struct [enet_rx_bd_ring_t](#)
Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)
- struct [enet_handle_t](#)
Defines the ENET handler structure. [More...](#)

Macros

- #define [ENET_BUFFDESCRIPTOR_RX_ERR_MASK](#)
Defines the receive error status flag mask.

Typedefs

- typedef void (*[enet_rx_alloc_callback_t](#))(ENET_Type *base, void *userData, uint8_t ringId)
Defines the ENET Rx memory buffer alloc function pointer.
- typedef void (*[enet_rx_free_callback_t](#))(ENET_Type *base, void *buffer, void *userData, uint8_t ringId)
Defines the ENET Rx memory buffer free function pointer.
- typedef void (*[enet_callback_t](#))(ENET_Type *base, enet_handle_t *handle, uint32_t ringId, [enet_event_t](#) event, [enet_frame_info_t](#) *frameInfo, void *userData)
ENET callback function.
- typedef void (*[enet_isr_ring_t](#))(ENET_Type *base, enet_handle_t *handle, uint32_t ringId)
Define interrupt IRQ handler.

Enumerations

- enum {
[kStatus_ENET_InitMemoryFail](#),
[kStatus_ENET_RxFrameError](#) = MAKE_STATUS(kStatusGroup_ENET, 1U),
[kStatus_ENET_RxFrameFail](#) = MAKE_STATUS(kStatusGroup_ENET, 2U),
[kStatus_ENET_RxFrameEmpty](#) = MAKE_STATUS(kStatusGroup_ENET, 3U),
[kStatus_ENET_RxFrameDrop](#) = MAKE_STATUS(kStatusGroup_ENET, 4U),
[kStatus_ENET_TxFrameOverLen](#) = MAKE_STATUS(kStatusGroup_ENET, 5U),
[kStatus_ENET_TxFrameBusy](#) = MAKE_STATUS(kStatusGroup_ENET, 6U),

```
kStatus_ENET_TxFrameFail = MAKE_STATUS(kStatusGroup_ENET, 7U) }
```

Defines the status return codes for transaction.

- enum `enet_mii_mode_t` {
`kENET_MiiMode` = 0U,
`kENET_RmiiMode` = 1U,
`kENET_RgmiiMode` = 2U }

Defines the MII/RMII/RGMII mode for data interface between the MAC and the PHY.

- enum `enet_mii_speed_t` {
`kENET_MiiSpeed10M` = 0U,
`kENET_MiiSpeed100M` = 1U,
`kENET_MiiSpeed1000M` = 2U }

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

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

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

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

Define the MII opcode for normal MDIO_CLAUSES_22 Frame.

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

Defines the read operation for the MII management frame.

- enum `enet_mii_extend_opcode` {
`kENET_MiiAddrWrite_C45` = 0U,
`kENET_MiiWriteFrame_C45` = 1U,
`kENET_MiiReadFrame_C45` = 3U }

Define the MII opcode for extended MDIO_CLAUSES_45 Frame.

- enum `enet_special_control_flag_t` {
`kENET_ControlFlowControlEnable` = 0x0001U,
`kENET_ControlRxPayloadCheckEnable` = 0x0002U,
`kENET_ControlRxPadRemoveEnable` = 0x0004U,
`kENET_ControlRxBroadCastRejectEnable` = 0x0008U,
`kENET_ControlMacAddrInsert` = 0x0010U,
`kENET_ControlStoreAndFwdDisable` = 0x0020U,
`kENET_ControlSMIPreambleDisable` = 0x0040U,
`kENET_ControlPromiscuousEnable` = 0x0080U,
`kENET_ControlMIILoopEnable` = 0x0100U,
`kENET_ControlVLANTagEnable` = 0x0200U,
`kENET_ControlSVLANEnable` = 0x0400U,
`kENET_ControlVLANUseSecondTag` = 0x0800U }

Defines a special configuration for ENET MAC controller.

- enum `enet_interrupt_enable_t` {

```

kENET_BabrInterrupt = ENET_EIR_BABR_MASK,
kENET_BabtInterrupt = ENET_EIR_BABT_MASK,
kENET_GraceStopInterrupt = ENET_EIR_GRA_MASK,
kENET_TxFrameInterrupt = ENET_EIR_TXF_MASK,
kENET_TxBufferInterrupt = ENET_EIR_TXB_MASK,
kENET_RxFrameInterrupt = ENET_EIR_RXF_MASK,
kENET_RxBufferInterrupt = ENET_EIR_RXB_MASK,
kENET_MiiInterrupt = ENET_EIR_MII_MASK,
kENET_EBusERInterrupt = ENET_EIR_EBERR_MASK,
kENET_LateCollisionInterrupt = ENET_EIR_LC_MASK,
kENET_RetryLimitInterrupt = ENET_EIR_RL_MASK,
kENET_UnderrunInterrupt = ENET_EIR_UN_MASK,
kENET_PayloadRxInterrupt = ENET_EIR_PLR_MASK,
kENET_WakeupInterrupt = ENET_EIR_WAKEUP_MASK,
kENET_RxFlush2Interrupt = ENET_EIR_RXFLUSH_2_MASK,
kENET_RxFlush1Interrupt = ENET_EIR_RXFLUSH_1_MASK,
kENET_RxFlush0Interrupt = ENET_EIR_RXFLUSH_0_MASK,
kENET_TxFrame2Interrupt = ENET_EIR_TXF2_MASK,
kENET_TxBuffer2Interrupt = ENET_EIR_TXB2_MASK,
kENET_RxFrame2Interrupt = ENET_EIR_RXF2_MASK,
kENET_RxBuffer2Interrupt = ENET_EIR_RXB2_MASK,
kENET_TxFrame1Interrupt = ENET_EIR_TXF1_MASK,
kENET_TxBuffer1Interrupt = ENET_EIR_TXB1_MASK,
kENET_RxFrame1Interrupt = ENET_EIR_RXF1_MASK,
kENET_RxBuffer1Interrupt = ENET_EIR_RXB1_MASK,
kENET_TsAvailInterrupt = ENET_EIR_TS_AVAIL_MASK,
kENET_TsTimerInterrupt = ENET_EIR_TS_TIMER_MASK }

```

List of interrupts supported by the peripheral.

- enum `enet_event_t` {
`kENET_RxEvent`,
`kENET_TxEvent`,
`kENET_ErrEvent`,
`kENET_WakeUpEvent`,
`kENET_TimeStampEvent`,
`kENET_TimeStampAvailEvent` }

Defines the common interrupt event for callback use.

- enum `enet_idle_slope_t` {

```

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

```

Defines certain idle slope for bandwidth fraction.

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

Defines the transmit accelerator configuration.

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

Defines the receive accelerator configuration.

Functions

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

Variables

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

Driver version

- #define `FSL_ENET_DRIVER_VERSION` (`MAKE_VERSION`(2, 5, 3))
Defines the driver version.

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

Defines the queue number.

- #define `ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK` 0x8000U
Empty bit mask.
- #define `ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK` 0x4000U
Software owner one mask.
- #define `ENET_BUFFDESCRIPTOR_RX_WRAP_MASK` 0x2000U
Next buffer descriptor is the start address.
- #define `ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask` 0x1000U
Software owner two mask.
- #define `ENET_BUFFDESCRIPTOR_RX_LAST_MASK` 0x0800U
Last BD of the frame mask.
- #define `ENET_BUFFDESCRIPTOR_RX_MISS_MASK` 0x0100U
Received because of the promiscuous mode.
- #define `ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK` 0x0080U
Broadcast packet mask.
- #define `ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK` 0x0040U
Multicast packet mask.
- #define `ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK` 0x0020U
Length violation mask.
- #define `ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK` 0x0010U
Non-octet aligned frame mask.
- #define `ENET_BUFFDESCRIPTOR_RX_CRC_MASK` 0x0004U
CRC error mask.
- #define `ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK` 0x0002U
FIFO overrun mask.
- #define `ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK` 0x0001U
Frame is truncated mask.

Control and status bit masks of the transmit buffer descriptor.

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

Defines some Ethernet parameters.

- #define `ENET_FRAME_MAX_FRAMELEN` 1518U
Default maximum Ethernet frame size without VLAN tag.
- #define `ENET_FRAME_VLAN_TAGLEN` 4U
Ethernet single VLAN tag size.

- #define **ENET_FRAME_CRC_LEN** 4U
CRC size in a frame.
- #define **ENET_FRAME_TX_LEN_LIMITATION**(x) (((x)->RCR & ENET_RCR_MAX_FL_MASK) >> ENET_RCR_MAX_FL_SHIFT) - **ENET_FRAME_CRC_LEN**)
- #define **ENET_FIFO_MIN_RX_FULL** 5U
ENET minimum receive FIFO full.
- #define **ENET_RX_MIN_BUFFERSIZE** 256U
ENET minimum buffer size.
- #define **ENET_PHY_MAXADDRESS** (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)
Maximum PHY address.
- #define **ENET_TX_INTERRUPT**
Enet Tx interrupt flag.
- #define **ENET_RX_INTERRUPT**
Enet Rx interrupt flag.
- #define **ENET_TS_INTERRUPT** ((uint32_t)kENET_TsTimerInterrupt | (uint32_t)kENET_TsAvailInterrupt)
Enet timestamp interrupt flag.
- #define **ENET_ERR_INTERRUPT**
Enet error interrupt flag.

Initialization and De-initialization

- void **ENET_GetDefaultConfig** (enet_config_t *config)
Gets the ENET default configuration structure.
- status_t **ENET_Up** (ENET_Type *base, enet_handle_t *handle, const enet_config_t *config, const enet_buffer_config_t *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)
Initializes the ENET module.
- status_t **ENET_Init** (ENET_Type *base, enet_handle_t *handle, const enet_config_t *config, const enet_buffer_config_t *bufferConfig, uint8_t *macAddr, uint32_t srcClock_Hz)
Initializes the ENET module.
- void **ENET_Down** (ENET_Type *base)
Stops the ENET module.
- void **ENET_Deinit** (ENET_Type *base)
Deinitializes the ENET module.
- static void **ENET_Reset** (ENET_Type *base)
Resets the ENET module.

MII interface operation

- void **ENET_SetMII** (ENET_Type *base, enet_mii_speed_t speed, enet_mii_duplex_t duplex)
Sets the ENET MII speed and duplex.
- void **ENET_SetSMI** (ENET_Type *base, uint32_t srcClock_Hz, bool isPreambleDisabled)
Sets the ENET SMI(serial management interface)- MII management interface.
- static bool **ENET_GetSMI** (ENET_Type *base)
Gets the ENET SMI- MII management interface configuration.
- static uint32_t **ENET_ReadSMIData** (ENET_Type *base)
Reads data from the PHY register through an SMI interface.
- void **ENET_StartSMIRead** (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, enet_mii_read_t operation)

- Starts an SMI (Serial Management Interface) read command.
- void [ENET_StartSMIWrite](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, [enet_mii_write_t](#) operation, uint32_t data)
Starts an SMI write command.
- void [ENET_StartExtC45SMIWriteReg](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg)
Starts the extended IEEE802.3 Clause 45 MDIO format SMI write register command.
- void [ENET_StartExtC45SMIWriteData](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg, uint32_t data)
Starts the extended IEEE802.3 Clause 45 MDIO format SMI write data command.
- void [ENET_StartExtC45SMIReadData](#) (ENET_Type *base, uint32_t phyAddr, uint32_t phyReg)
Starts the extended IEEE802.3 Clause 45 MDIO format SMI read data command.
- static void [ENET_SetRGMIIIClockDelay](#) (ENET_Type *base, bool txEnabled, bool rxEnabled)
Control the usage of the delayed tx/rx RGMII clock.

MAC Address Filter

- void [ENET_SetMacAddr](#) (ENET_Type *base, uint8_t *macAddr)
Sets the ENET module Mac address.
- void [ENET_GetMacAddr](#) (ENET_Type *base, uint8_t *macAddr)
Gets the ENET module Mac address.
- void [ENET_AddMulticastGroup](#) (ENET_Type *base, uint8_t *address)
Adds the ENET device to a multicast group.
- void [ENET_LeaveMulticastGroup](#) (ENET_Type *base, uint8_t *address)
Moves the ENET device from a multicast group.

Other basic operation

- static void [ENET_ActiveRead](#) (ENET_Type *base)
Activates frame reception for multiple rings.
- static void [ENET_EnableSleepMode](#) (ENET_Type *base, bool enable)
Enables/disables the MAC to enter sleep mode.
- static void [ENET_GetAccelFunction](#) (ENET_Type *base, uint32_t *txAccelOption, uint32_t *rxAccelOption)
Gets ENET transmit and receive accelerator functions from MAC controller.

Interrupts.

- static void [ENET_EnableInterrupts](#) (ENET_Type *base, uint32_t mask)
Enables the ENET interrupt.
- static void [ENET_DisableInterrupts](#) (ENET_Type *base, uint32_t mask)
Disables the ENET interrupt.
- static uint32_t [ENET_GetInterruptStatus](#) (ENET_Type *base)
Gets the ENET interrupt status flag.
- static void [ENET_ClearInterruptStatus](#) (ENET_Type *base, uint32_t mask)
Clears the ENET interrupt events status flag.
- void [ENET_SetRxISRHandler](#) (ENET_Type *base, [enet_isr_ring_t](#) ISRHandler)
Set the second level Rx IRQ handler.
- void [ENET_SetTxISRHandler](#) (ENET_Type *base, [enet_isr_ring_t](#) ISRHandler)
Set the second level Tx IRQ handler.
- void [ENET_SetErrISRHandler](#) (ENET_Type *base, [enet_isr_t](#) ISRHandler)
Set the second level Err IRQ handler.

Transactional operation

- void [ENET_SetCallback](#) (enet_handle_t *handle, [enet_callback_t](#) callback, void *userData)
Sets the callback function.
- void [ENET_GetRxErrBeforeReadFrame](#) (enet_handle_t *handle, [enet_data_error_stats_t](#) *eError-Static, uint8_t ringId)
Gets the error statistics of a received frame for ENET specified ring.
- void [ENET_GetStatistics](#) (ENET_Type *base, [enet_transfer_stats_t](#) *statistics)
Gets statistical data in transfer.
- [status_t](#) [ENET_GetRxFrameSize](#) (enet_handle_t *handle, uint32_t *length, uint8_t ringId)
Gets the size of the read frame for specified ring.
- [status_t](#) [ENET_ReadFrame](#) (ENET_Type *base, enet_handle_t *handle, uint8_t *data, uint32_t length, uint8_t ringId, uint32_t *ts)
Reads a frame from the ENET device.
- [status_t](#) [ENET_SendFrame](#) (ENET_Type *base, enet_handle_t *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)
Transmits an ENET frame for specified ring.
- [status_t](#) [ENET_SetTxReclaim](#) (enet_handle_t *handle, bool isEnabled, uint8_t ringId)
Enable or disable tx descriptors reclaim mechanism.
- void [ENET_ReclaimTxDescriptor](#) (ENET_Type *base, enet_handle_t *handle, uint8_t ringId)
Reclaim tx descriptors.
- [status_t](#) [ENET_GetRxBuffer](#) (ENET_Type *base, enet_handle_t *handle, void **buffer, uint32_t *length, uint8_t ringId, bool *isLastBuff, uint32_t *ts)
Get a receive buffer pointer of the ENET device for specified ring.
- void [ENET_ReleaseRxBuffer](#) (ENET_Type *base, enet_handle_t *handle, void *buffer, uint8_t ringId)
Release receive buffer descriptor to DMA.
- [status_t](#) [ENET_GetRxFrame](#) (ENET_Type *base, enet_handle_t *handle, [enet_rx_frame_struct_t](#) *rxFrame, uint8_t ringId)
Receives one frame in specified BD ring with zero copy.
- [status_t](#) [ENET_StartTxFrame](#) (ENET_Type *base, enet_handle_t *handle, [enet_tx_frame_struct_t](#) *txFrame, uint8_t ringId)
Sends one frame in specified BD ring with zero copy.
- [status_t](#) [ENET_SendFrameZeroCopy](#) (ENET_Type *base, enet_handle_t *handle, const uint8_t *data, uint32_t length, uint8_t ringId, bool tsFlag, void *context)
Transmits an ENET frame for specified ring with zero-copy.
- void [ENET_TransmitIRQHandler](#) (ENET_Type *base, enet_handle_t *handle, uint32_t ringId)
The transmit IRQ handler.
- void [ENET_ReceiveIRQHandler](#) (ENET_Type *base, enet_handle_t *handle, uint32_t ringId)
The receive IRQ handler.
- void [ENET_CommonFrame1IRQHandler](#) (ENET_Type *base)
the common IRQ handler for the tx/rx irq handler.
- void [ENET_CommonFrame2IRQHandler](#) (ENET_Type *base)
the common IRQ handler for the tx/rx irq handler.
- void [ENET_ErrorIRQHandler](#) (ENET_Type *base, enet_handle_t *handle)
Some special IRQ handler including the error, mii, wakeup irq handler.
- void [ENET_Ptp1588IRQHandler](#) (ENET_Type *base)
the common IRQ handler for the 1588 irq handler.
- void [ENET_CommonFrame0IRQHandler](#) (ENET_Type *base)
the common IRQ handler for the tx/rx/error etc irq handler.

28.4 Data Structure Documentation

28.4.1 struct enet_rx_bd_struct_t

Data Fields

- uint16_t [length](#)
Buffer descriptor data length.
- uint16_t [control](#)
Buffer descriptor control and status.
- uint8_t * [buffer](#)
Data buffer pointer.

Field Documentation

(1) uint16_t enet_rx_bd_struct_t::length

(2) uint16_t enet_rx_bd_struct_t::control

(3) uint8_t* enet_rx_bd_struct_t::buffer

28.4.2 struct enet_tx_bd_struct_t

Data Fields

- uint16_t [length](#)
Buffer descriptor data length.
- uint16_t [control](#)
Buffer descriptor control and status.
- uint8_t * [buffer](#)
Data buffer pointer.

Field Documentation

(1) uint16_t enet_tx_bd_struct_t::length

(2) uint16_t enet_tx_bd_struct_t::control

(3) uint8_t* enet_tx_bd_struct_t::buffer

28.4.3 struct enet_data_error_stats_t

Data Fields

- uint32_t [statsRxLenGreaterErr](#)
Receive length greater than RCR[MAX_FL].
- uint32_t [statsRxAlignErr](#)
Receive non-octet alignment/.
- uint32_t [statsRxFcsErr](#)

- *Receive CRC error.*
uint32_t statsRxOverRunErr
- *Receive over run.*
uint32_t statsRxTruncateErr
- *Receive truncate.*

Field Documentation

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

28.4.4 struct enet_rx_frame_error_t

Data Fields

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

Field Documentation

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

28.4.5 struct enet_transfer_stats_t

Data Fields

- uint32_t statsRxFrameCount
Rx frame number.

- uint32_t [statsRxFrameOk](#)
Good Rx frame number.
- uint32_t [statsRxCrcErr](#)
Rx frame number with CRC error.
- uint32_t [statsRxAlignErr](#)
Rx frame number with alignment error.
- uint32_t [statsRxDropInvalidSFD](#)
Dropped frame number due to invalid SFD.
- uint32_t [statsRxFifoOverflowErr](#)
Rx FIFO overflow count.
- uint32_t [statsTxFrameCount](#)
Tx frame number.
- uint32_t [statsTxFrameOk](#)
Good Tx frame number.
- uint32_t [statsTxCrcAlignErr](#)
The transmit frame is error.
- uint32_t [statsTxFifoUnderRunErr](#)
Tx FIFO underrun count.

Field Documentation

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

28.4.6 struct enet_frame_info_t

Data Fields

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

28.4.7 struct enet_tx_dirty_ring_t

Data Fields

- [enet_frame_info_t](#) * [txDirtyBase](#)
Dirty buffer descriptor base address pointer.
- [uint16_t](#) [txGenIdx](#)
tx generate index.
- [uint16_t](#) [txConsumIdx](#)
tx consume index.
- [uint16_t](#) [txRingLen](#)
tx ring length.
- [bool](#) [isFull](#)
tx ring is full flag.

Field Documentation

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

28.4.8 struct enet_buffer_config_t

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

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

Data Fields

- [uint16_t](#) [rxBdNumber](#)

- *Receive buffer descriptor number.*
uint16_t [rxBdNumber](#)
- *Transmit buffer descriptor number.*
uint16_t [rxBuffSizeAlign](#)
- *Aligned receive data buffer size.*
uint16_t [txBuffSizeAlign](#)
- *Aligned transmit data buffer size.*
volatile [enet_rx_bd_struct_t](#) * [rxBdStartAddrAlign](#)
- *Aligned receive buffer descriptor start address: should be non-cacheable.*
volatile [enet_tx_bd_struct_t](#) * [txBdStartAddrAlign](#)
- *Aligned transmit buffer descriptor start address: should be non-cacheable.*
uint8_t * [rxBufferAlign](#)
- *Receive data buffer start address.*
uint8_t * [txBufferAlign](#)
- *Transmit data buffer start address.*
bool [rxMaintainEnable](#)
- *Receive buffer cache maintain.*
bool [txMaintainEnable](#)
- *Transmit buffer cache maintain.*
[enet_frame_info_t](#) * [txFrameInfo](#)
- *Transmit frame information start address.*

Field Documentation

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

28.4.9 struct enet_intcoalesce_config_t

Data Fields

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

Field Documentation

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

28.4.10 struct enet_avb_config_t

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

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

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

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

Data Fields

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

Field Documentation

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

28.4.11 struct enet_config_t

Note:

1. macSpecialConfig is used for a special control configuration, A logical OR of "enet_special_control_flag_t". For a special configuration for MAC, set this parameter to 0.
2. txWatermark is used for a cut-through operation. It is in steps of 64 bytes: 0/1 - 64 bytes written to TX FIFO before transmission of a frame begins. 2 - 128 bytes written to TX FIFO 3 - 192 bytes written to TX FIFO The maximum of txWatermark is 0x2F - 4032 bytes written to TX FIFO txWatermark allows minimizing the transmit latency to set the txWatermark to 0 or 1 or for larger bus access latency 3 or larger due to contention for the system bus.
3. rxFifoFullThreshold is similar to the txWatermark for cut-through operation in RX. It is in 64-bit words. The minimum is ENET_FIFO_MIN_RX_FULL and the maximum is 0xFF. If the end of the frame is stored in FIFO and the frame size is smaller than the txWatermark, the frame is still transmitted. The rule is the same for rxFifoFullThreshold in the receive direction.
4. When "kENET_ControlFlowControlEnable" is set in the macSpecialConfig, ensure that the pauseDuration, rxFifoEmptyThreshold, and rxFifoStatEmptyThreshold are set for flow control enabled case.
5. When "kENET_ControlStoreAndFwdDisabled" is set in the macSpecialConfig, ensure that the rxFifoFullThreshold and txFifoWatermark are set for store and forward disable.
6. The rxAccelerConfig and txAccelerConfig default setting with 0 - accelerator are disabled. The "enet_tx_accelerator_t" and "enet_rx_accelerator_t" are recommended to be used to enable the transmit and receive accelerator. After the accelerators are enabled, the store and forward feature should be enabled. As a result, kENET_ControlStoreAndFwdDisabled should not be set.
7. The intCoalesceCfg can be used in the rx or tx enabled cases to decrease the CPU loading.

Data Fields

- uint32_t [macSpecialConfig](#)
Mac special configuration.
- uint32_t [interrupt](#)
Mac interrupt source.
- uint16_t [rxMaxFrameLen](#)
Receive maximum frame length.
- [enet_mii_mode_t](#) [miiMode](#)
MII mode.
- [enet_mii_speed_t](#) [miiSpeed](#)
MII Speed.
- [enet_mii_duplex_t](#) [miiDuplex](#)
MII duplex.
- uint8_t [rxAccelerConfig](#)
Receive accelerator, A logical OR of "enet_rx_accelerator_t".
- uint8_t [txAccelerConfig](#)
Transmit accelerator, A logical OR of "enet_rx_accelerator_t".
- uint16_t [pauseDuration](#)
For flow control enabled case: Pause duration.
- uint8_t [rxFifoEmptyThreshold](#)

For flow control enabled case: when RX FIFO level reaches this value, it makes MAC generate XOFF pause frame.

- `uint8_t rxFifoStatEmptyThreshold`

For flow control enabled case: number of frames in the receive FIFO, independent of size, that can be accept.

- `uint8_t rxFifoFullThreshold`

For store and forward disable case, the data required in RX FIFO to notify the MAC receive ready status.

- `uint8_t txFifoWatermark`

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

- `enet_intcoalesce_config_t * intCoalesceCfg`

If the interrupt coalesce is not required in the ring n(0,1,2), please set to NULL.

- `uint8_t ringNum`

Number of used rings.

- `enet_rx_alloc_callback_t rxBuffAlloc`

Callback function to alloc memory, must be provided for zero-copy Rx.

- `enet_rx_free_callback_t rxBuffFree`

Callback function to free memory, must be provided for zero-copy Rx.

- `enet_callback_t callback`

General callback function.

- `void * userData`

Callback function parameter.

Field Documentation

(1) `uint32_t enet_config_t::macSpecialConfig`

A logical OR of "enet_special_control_flag_t".

(2) `uint32_t enet_config_t::interrupt`

A logical OR of "enet_interrupt_enable_t".

(3) `uint16_t enet_config_t::rxMaxFrameLen`

(4) `enet_mii_mode_t enet_config_t::miiMode`

(5) `enet_mii_speed_t enet_config_t::miiSpeed`

(6) `enet_mii_duplex_t enet_config_t::miiDuplex`

(7) `uint8_t enet_config_t::rxAccelerConfig`

(8) `uint8_t enet_config_t::txAccelerConfig`

(9) `uint16_t enet_config_t::pauseDuration`

(10) `uint8_t enet_config_t::rxFifoEmptyThreshold`

(11) `uint8_t enet_config_t::rxFifoStatEmptyThreshold`

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

(12) `uint8_t enet_config_t::rxFifoFullThreshold`

(13) `uint8_t enet_config_t::txFifoWatermark`

(14) `enet_intcoalesce_config_t* enet_config_t::intCoalesceCfg`

(15) `uint8_t enet_config_t::ringNum`

default with 1 – single ring.

(16) `enet_rx_alloc_callback_t enet_config_t::rxBuffAlloc`

(17) `enet_rx_free_callback_t enet_config_t::rxBuffFree`

(18) `enet_callback_t enet_config_t::callback`

(19) `void* enet_config_t::userData`

28.4.12 struct `enet_tx_bd_ring_t`

Data Fields

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

Field Documentation

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

(2) `uint16_t enet_tx_bd_ring_t::txGenIdx`

(3) `uint16_t enet_tx_bd_ring_t::txConsumeIdx`

(4) `volatile uint16_t enet_tx_bd_ring_t::txDescUsed`

(5) `uint16_t enet_tx_bd_ring_t::txRingLen`

28.4.13 struct enet_rx_bd_ring_t

Data Fields

- volatile [enet_rx_bd_struct_t](#) * [rxBdBase](#)
Buffer descriptor base address pointer.
- uint16_t [rxGenIdx](#)
The current available receive buffer descriptor pointer.
- uint16_t [rxRingLen](#)
Receive ring length.

Field Documentation

(1) volatile [enet_rx_bd_struct_t](#)* [enet_rx_bd_ring_t::rxBdBase](#)

(2) uint16_t [enet_rx_bd_ring_t::rxGenIdx](#)

(3) uint16_t [enet_rx_bd_ring_t::rxRingLen](#)

28.4.14 struct _enet_handle

Data Fields

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

- uint8_t `multicastCount` [64]
Multicast collisions counter.

Field Documentation

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

28.5 Macro Definition Documentation

28.5.1 `#define FSL_ENET_DRIVER_VERSION (MAKE_VERSION(2, 5, 3))`

28.5.2 `#define ENET_BUFFDESCRIPTOR_RX_EMPTY_MASK 0x8000U`

28.5.3 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER1_MASK 0x4000U`

28.5.4 `#define ENET_BUFFDESCRIPTOR_RX_WRAP_MASK 0x2000U`

28.5.5 `#define ENET_BUFFDESCRIPTOR_RX_SOFTOWNER2_Mask 0x1000U`

28.5.6 `#define ENET_BUFFDESCRIPTOR_RX_LAST_MASK 0x0800U`

28.5.7 `#define ENET_BUFFDESCRIPTOR_RX_MISS_MASK 0x0100U`

28.5.8 #define ENET_BUFFDESCRIPTOR_RX_BROADCAST_MASK 0x0080U

28.5.9 #define ENET_BUFFDESCRIPTOR_RX_MULTICAST_MASK 0x0040U

28.5.10 #define ENET_BUFFDESCRIPTOR_RX_LENVIOLATE_MASK 0x0020U

28.5.11 #define ENET_BUFFDESCRIPTOR_RX_NOOCTET_MASK 0x0010U

28.5.12 #define ENET_BUFFDESCRIPTOR_RX_CRC_MASK 0x0004U

28.5.13 #define ENET_BUFFDESCRIPTOR_RX_OVERRUN_MASK 0x0002U

28.5.14 #define ENET_BUFFDESCRIPTOR_RX_TRUNC_MASK 0x0001U

28.5.15 #define ENET_BUFFDESCRIPTOR_TX_READY_MASK 0x8000U

28.5.16 #define ENET_BUFFDESCRIPTOR_TX_SOFTWENER1_MASK 0x4000U

28.5.17 #define ENET_BUFFDESCRIPTOR_TX_WRAP_MASK 0x2000U

28.5.18 #define ENET_BUFFDESCRIPTOR_TX_SOFTWENER2_MASK 0x1000U

28.5.19 #define ENET_BUFFDESCRIPTOR_TX_LAST_MASK 0x0800U

28.5.20 #define ENET_BUFFDESCRIPTOR_TX_TRANMITCRC_MASK 0x0400U

28.5.21 #define ENET_BUFFDESCRIPTOR_RX_ERR_MASK

Value:

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

28.5.22 #define ENET_FRAME_MAX_FRAMELEN 1518U

28.5.23 #define ENET_FRAME_VLAN_TAGLEN 4U

28.5.24 #define ENET_FRAME_CRC_LEN 4U

28.5.25 #define ENET_FIFO_MIN_RX_FULL 5U

28.5.26 #define ENET_RX_MIN_BUFFERSIZE 256U

28.5.27 #define ENET_PHY_MAXADDRESS (ENET_MMFR_PA_MASK >> ENET_MMFR_PA_SHIFT)

28.5.28 #define ENET_TX_INTERRUPT

Value:

```
((uint32_t)kENET_TxFrameInterrupt | (uint32_t)
    kENET_TxBufferInterrupt | (uint32_t)
    kENET_TxFrame1Interrupt | \
    (uint32_t)kENET_TxBuffer1Interrupt | (uint32_t)
    kENET_TxFrame2Interrupt |
    (uint32_t)kENET_TxBuffer2Interrupt) \
```

28.5.29 #define ENET_RX_INTERRUPT

Value:

```
((uint32_t)kENET_RxFrameInterrupt | (uint32_t)
    kENET_RxBufferInterrupt | (uint32_t)
    kENET_RxFrame1Interrupt | \
    (uint32_t)kENET_RxBuffer1Interrupt | (uint32_t)
    kENET_RxFrame2Interrupt |
    (uint32_t)kENET_RxBuffer2Interrupt) \
```

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

28.5.31 #define ENET_ERR_INTERRUPT

Value:

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

28.6 Typedef Documentation

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

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

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

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

28.7 Enumeration Type Documentation

28.7.1 anonymous enum

Enumerator

kStatus_ENET_InitMemoryFail Init fails since buffer memory is not enough.

kStatus_ENET_RxFrameError A frame received but data error happen.

kStatus_ENET_RxFrameFail Failed to receive a frame.

kStatus_ENET_RxFrameEmpty No frame arrive.

kStatus_ENET_RxFrameDrop Rx frame is dropped since no buffer memory.

kStatus_ENET_TxFrameOverLen Tx frame over length.

kStatus_ENET_TxFrameBusy Tx buffer descriptors are under process.

kStatus_ENET_TxFrameFail Transmit frame fail.

28.7.2 enum enet_mii_mode_t

Enumerator

kENET_MiiMode MII mode for data interface.

kENET_RmiiMode RMII mode for data interface.

kENET_RgmiiMode RGMII mode for data interface.

28.7.3 enum enet_mii_speed_t

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

Enumerator

kENET_MiiSpeed10M Speed 10 Mbps.
kENET_MiiSpeed100M Speed 100 Mbps.
kENET_MiiSpeed1000M Speed 1000M bps.

28.7.4 enum enet_mii_duplex_t

Enumerator

kENET_MiiHalfDuplex Half duplex mode.
kENET_MiiFullDuplex Full duplex mode.

28.7.5 enum enet_mii_write_t

Enumerator

kENET_MiiWriteNoCompliant Write frame operation, but not MII-compliant.
kENET_MiiWriteValidFrame Write frame operation for a valid MII management frame.

28.7.6 enum enet_mii_read_t

Enumerator

kENET_MiiReadValidFrame Read frame operation for a valid MII management frame.
kENET_MiiReadNoCompliant Read frame operation, but not MII-compliant.

28.7.7 enum enet_mii_extend_opcode

Enumerator

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

28.7.8 enum enet_special_control_flag_t

These control flags are provided for special user requirements. Normally, these control flags are unused for ENET initialization. For special requirements, set the flags to macSpecialConfig in the [enet_config_t](#).

The `kENET_ControlStoreAndFwdDisable` is used to disable the FIFO store and forward. FIFO store and forward means that the FIFO read/send is started when a complete frame is stored in TX/RX FIFO. If this flag is set, configure `rxFifoFullThreshold` and `txFifoWatermark` in the [enet_config_t](#).

Enumerator

kENET_ControlFlowControlEnable Enable ENET flow control: pause frame.
kENET_ControlRxPayloadCheckEnable Enable ENET receive payload length check.
kENET_ControlRxPadRemoveEnable Padding is removed from received frames.
kENET_ControlRxBroadcastRejectEnable Enable broadcast frame reject.
kENET_ControlMacAddrInsert Enable MAC address insert.
kENET_ControlStoreAndFwdDisable Enable FIFO store and forward.
kENET_ControlSMIPreambleDisable Enable SMI preamble.
kENET_ControlPromiscuousEnable Enable promiscuous mode.
kENET_ControlMIILoopEnable Enable ENET MII loop back.
kENET_ControlVLANTagEnable Enable normal VLAN (single vlan tag).
kENET_ControlSVLANEnable Enable S-VLAN.
kENET_ControlVLANUseSecondTag Enable extracting the second vlan tag for further processing.

28.7.9 enum enet_interrupt_enable_t

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

Enumerator

kENET_BabrInterrupt Babbling receive error interrupt source.
kENET_BabtInterrupt Babbling transmit error interrupt source.
kENET_GraceStopInterrupt Graceful stop complete interrupt source.
kENET_TxFrameInterrupt TX FRAME interrupt source.
kENET_TxBufferInterrupt TX BUFFER interrupt source.
kENET_RxFrameInterrupt RX FRAME interrupt source.
kENET_RxBufferInterrupt RX BUFFER interrupt source.
kENET_MiiInterrupt MII interrupt source.
kENET_EBusERInterrupt Ethernet bus error interrupt source.
kENET_LateCollisionInterrupt Late collision interrupt source.
kENET_RetryLimitInterrupt Collision Retry Limit interrupt source.
kENET_UnderrunInterrupt Transmit FIFO underrun interrupt source.
kENET_PayloadRxInterrupt Payload Receive error interrupt source.
kENET_WakeupInterrupt WAKEUP interrupt source.
kENET_RxFlush2Interrupt Rx DMA ring2 flush indication.
kENET_RxFlush1Interrupt Rx DMA ring1 flush indication.
kENET_RxFlush0Interrupt RX DMA ring0 flush indication.
kENET_TxFrame2Interrupt Tx frame interrupt for Tx ring/class 2.
kENET_TxBuffer2Interrupt Tx buffer interrupt for Tx ring/class 2.

kENET_RxFrame2Interrupt Rx frame interrupt for Rx ring/class 2.
kENET_RxBuffer2Interrupt Rx buffer interrupt for Rx ring/class 2.
kENET_TxFrame1Interrupt Tx frame interrupt for Tx ring/class 1.
kENET_TxBuffer1Interrupt Tx buffer interrupt for Tx ring/class 1.
kENET_RxFrame1Interrupt Rx frame interrupt for Rx ring/class 1.
kENET_RxBuffer1Interrupt Rx buffer interrupt for Rx ring/class 1.
kENET_TsAvailInterrupt TS AVAIL interrupt source for PTP.
kENET_TsTimerInterrupt TS WRAP interrupt source for PTP.

28.7.10 enum enet_event_t

Enumerator

kENET_RxEvent Receive event.
kENET_TxEvent Transmit event.
kENET_ErrEvent Error event: BABR/BABT/EBERR/LC/RL/UN/PLR .
kENET_WakeUpEvent Wake up from sleep mode event.
kENET_TimeStampEvent Time stamp event.
kENET_TimeStampAvailEvent Time stamp available event.

28.7.11 enum enet_idle_slope_t

Enumerator

kENET_IdleSlope1 The bandwidth fraction is about 0.002.
kENET_IdleSlope2 The bandwidth fraction is about 0.003.
kENET_IdleSlope4 The bandwidth fraction is about 0.008.
kENET_IdleSlope8 The bandwidth fraction is about 0.02.
kENET_IdleSlope16 The bandwidth fraction is about 0.03.
kENET_IdleSlope32 The bandwidth fraction is about 0.06.
kENET_IdleSlope64 The bandwidth fraction is about 0.11.
kENET_IdleSlope128 The bandwidth fraction is about 0.20.
kENET_IdleSlope256 The bandwidth fraction is about 0.33.
kENET_IdleSlope384 The bandwidth fraction is about 0.43.
kENET_IdleSlope512 The bandwidth fraction is about 0.50.
kENET_IdleSlope640 The bandwidth fraction is about 0.56.
kENET_IdleSlope768 The bandwidth fraction is about 0.60.
kENET_IdleSlope896 The bandwidth fraction is about 0.64.
kENET_IdleSlope1024 The bandwidth fraction is about 0.67.
kENET_IdleSlope1152 The bandwidth fraction is about 0.69.
kENET_IdleSlope1280 The bandwidth fraction is about 0.71.
kENET_IdleSlope1408 The bandwidth fraction is about 0.73.
kENET_IdleSlope1536 The bandwidth fraction is about 0.75.

28.7.12 enum enet_tx_accelerator_t

Enumerator

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

28.7.13 enum enet_rx_accelerator_t

Enumerator

kENET_RxAccelPadRemoveEnabled Padding removal for short IP frames.
kENET_RxAccelIpCheckEnabled Discard with wrong IP header checksum.
kENET_RxAccelProtoCheckEnabled Discard with wrong protocol checksum.
kENET_RxAccelMacCheckEnabled Discard with Mac layer errors.
kENET_RxAccelIsShift16Enabled Receive FIFO shift-16.

28.8 Function Documentation

28.8.1 uint32_t ENET_GetInstance (ENET_Type * *base*)

Parameters

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

Returns

ENET instance.

28.8.2 void ENET_GetDefaultConfig (enet_config_t * *config*)

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

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

Parameters

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

28.8.3 status_t ENET_Up (ENET_Type * *base*, enet_handle_t * *handle*, const enet_config_t * *config*, const enet_buffer_config_t * *bufferConfig*, uint8_t * *macAddr*, uint32_t *srcClock_Hz*)

This function initializes the module with the ENET configuration.

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET_Up\(\)](#).

Parameters

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

<i>macAddr</i>	ENET mac address of Ethernet device. This MAC address should be provided.
<i>srcClock_Hz</i>	The internal module clock source for MII clock.

Return values

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

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

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

Note

ENET has two buffer descriptors legacy buffer descriptors and enhanced IEEE 1588 buffer descriptors. The legacy descriptor is used by default. To use the IEEE 1588 feature, use the enhanced IEEE 1588 buffer descriptor by defining "ENET_ENHANCEDBUFFERDESCRIPTOR_MODE" and calling ENET_Ptp1588Configure() to configure the 1588 feature and related buffers after calling [ENET_Init\(\)](#).

Parameters

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

<i>macAddr</i>	ENET mac address of Ethernet device. This MAC address should be provided.
<i>srcClock_Hz</i>	The internal module clock source for MII clock.

Return values

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

28.8.5 void ENET_Down (ENET_Type * *base*)

This function disables the ENET module.

Parameters

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

28.8.6 void ENET_Deinit (ENET_Type * *base*)

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

Parameters

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

28.8.7 static void ENET_Reset (ENET_Type * *base*) [inline], [static]

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

Parameters

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

28.8.8 void ENET_SetMII (ENET_Type * *base*, enet_mii_speed_t *speed*, enet_mii_duplex_t *duplex*)

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

Parameters

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

28.8.9 void ENET_SetSMI (ENET_Type * *base*, uint32_t *srcClock_Hz*, bool *isPreambleDisabled*)

Parameters

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

28.8.10 static bool ENET_GetSMI (ENET_Type * *base*) [inline], [static]

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

Parameters

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

Returns

The SMI setup status true or false.

28.8.11 static uint32_t ENET_ReadSMIData (ENET_Type * *base*) [inline], [static]

Parameters

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

Returns

The data read from PHY

28.8.12 void ENET_StartSMIRead (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, enet_mii_read_t *operation*)

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

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

28.8.13 void ENET_StartSMIWrite (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, enet_mii_write_t *operation*, uint32_t *data*)

Used for standard IEEE802.3 MDIO Clause 22 format.

Parameters

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

28.8.14 void ENET_StartExtC45SMIWriteReg (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*)

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): $\text{phyReg} = (\text{devaddr} \ll 16) \mid \text{regAddr}$.

28.8.15 void ENET_StartExtC45SMIWriteData (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*, uint32_t *data*)

After writing MMFR register, we need to check whether the transmission is over. This is an example for whole procedure of clause 45 MDIO write.

```

*   ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*   ENET_StartExtC45SMIWriteReg(base, phyAddr, phyReg);
*   while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*   {
*   }
*   ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*   ENET_StartExtC45SMIWriteData(base, phyAddr, phyReg, data);
*   while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*   {
*   }
*   ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*

```

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): $\text{phyReg} = (\text{devaddr} \ll 16) \mid \text{regAddr}$.
<i>data</i>	The data written to PHY.

28.8.16 void ENET_StartExtC45SMIReadData (ENET_Type * *base*, uint32_t *phyAddr*, uint32_t *phyReg*)

After writing MMFR register, we need to check whether the transmission is over. This is an example for whole procedure of clause 45 MDIO read.

```

*   uint32_t data;
*   ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);

```

```

*   ENET_StartExtC45SMIWriteReg(base, phyAddr, phyReg);
*   while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*   {
*   }
*   ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*   ENET_StartExtC45SMIReadData(base, phyAddr, phyReg);
*   while ((ENET_GetInterruptStatus(base) & ENET_EIR_MII_MASK) == 0U)
*   {
*   }
*   ENET_ClearInterruptStatus(base, ENET_EIR_MII_MASK);
*   data = ENET_ReadSMIData(base);
*

```

Parameters

<i>base</i>	ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register. For MDIO IEEE802.3 Clause 45, the phyReg is a 21-bits combination of the devaddr (5 bits device address) and the regAddr (16 bits phy register): phyReg = (devaddr << 16) regAddr.

28.8.17 static void ENET_SetRGMIIIClockDelay (ENET_Type * *base*, bool *txEnabled*, bool *rxEnabled*) [inline], [static]

Parameters

<i>base</i>	ENET peripheral base address.
<i>txEnabled</i>	Enable or disable to generate the delayed version of RGMII_TXC.
<i>rxEnabled</i>	Enable or disable to use the delayed version of RGMII_RXC.

28.8.18 void ENET_SetMacAddr (ENET_Type * *base*, uint8_t * *macAddr*)

Parameters

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

28.8.19 void ENET_GetMacAddr (ENET_Type * *base*, uint8_t * *macAddr*)

Parameters

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

28.8.20 void ENET_AddMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Parameters

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

28.8.21 void ENET_LeaveMulticastGroup (ENET_Type * *base*, uint8_t * *address*)

Parameters

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

**28.8.22 static void ENET_ActiveRead (ENET_Type * *base*) [inline],
[static]**

This function is to active the enet read process.

Note

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

Parameters

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

28.8.23 `static void ENET_EnableSleepMode (ENET_Type * base, bool enable)`
`[inline], [static]`

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

Parameters

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

28.8.24 static void ENET_GetAccelFunction (ENET_Type * *base*, uint32_t * *txAccelOption*, uint32_t * *rxAccelOption*) [inline], [static]

Parameters

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

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

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

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

Parameters

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

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

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


```

*  ENET_DisableInterrupts (ENET, kENET_TxFrameInterrupt |
*  kENET_RxFrameInterrupt);

```

Parameters

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

28.8.27 static uint32_t ENET_GetInterruptStatus (ENET_Type * *base*) [inline], [static]

Parameters

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

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration [enet_interrupt_enable_t](#).

28.8.28 static void ENET_ClearInterruptStatus (ENET_Type * *base*, uint32_t *mask*) [inline], [static]

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

```

*  ENET_ClearInterruptStatus (ENET,
*  kENET_TxFrameInterrupt | kENET_RxFrameInterrupt);

```

Parameters

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

<i>mask</i>	ENET interrupt source to be cleared. This is the logical OR of members of the enumeration enet_interrupt_enable_t .
-------------	---

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

Parameters

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

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

Parameters

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

28.8.31 void ENET_SetErrISRHandler (ENET_Type * *base*, enet_isr_t *ISRHandler*)

Parameters

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

28.8.32 void ENET_SetCallback (enet_handle_t * *handle*, enet_callback_t *callback*, void * *userData*)

Deprecated Do not use this function. It has been superseded by the config param in [ENET_Init](#). This API is provided for the application callback required case when ENET interrupt is enabled. This API should be called after calling ENET_Init.

Parameters

<i>handle</i>	ENET handler pointer. Should be provided by application.
<i>callback</i>	The ENET callback function.
<i>userData</i>	The callback function parameter.

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

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

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

Parameters

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

28.8.34 void ENET_GetStatistics (ENET_Type * *base*, enet_transfer_stats_t * *statistics*)

Parameters

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

28.8.35 `status_t ENET_GetRxFrameSize (enet_handle_t * handle, uint32_t * length, uint8_t ringId)`

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

Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling `ENET_GetRxFrameSize`, `ENET_ReadFrame()` should be called to receive frame and update the BD if the result is not "kStatus_ENET_RxFrameEmpty".

Parameters

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

Return values

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

28.8.36 `status_t ENET_ReadFrame (ENET_Type * base, enet_handle_t * handle, uint8_t * data, uint32_t length, uint8_t ringId, uint32_t * ts)`

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

Note

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

```
*      uint32_t length;
*      enet_handle_t g_handle;
*      Comments: Get the received frame size firstly.
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
```

```

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

```

Parameters

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

Returns

The execute status, successful or failure.

28.8.37 `status_t ENET_SendFrame (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, bool tsFlag, void * context)`

Note

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

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler pointer. This is the same handler pointer used in the ENET_Init.
<i>data</i>	The data buffer provided by user to send.
<i>length</i>	The length of the data to send.
<i>ringId</i>	The ring index or ring number.
<i>tsFlag</i>	Timestamp enable flag.
<i>context</i>	Used by user to handle some events after transmit over.

Return values

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

28.8.38 status_t ENET_SetTxReclaim (enet_handle_t * handle, bool isEnabled, uint8_t ringId)

Note

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

Parameters

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

Return values

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

28.8.39 void ENET_ReclaimTxDescriptor (ENET_Type * *base*, enet_handle_t * *handle*, uint8_t *ringld*)

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

Parameters

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

28.8.40 `status_t ENET_GetRxBuffer (ENET_Type * base, enet_handle_t * handle, void ** buffer, uint32_t * length, uint8_t ringId, bool * isLastBuff, uint32_t * ts)`

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

This function can get the data address which stores frame. Then can analyze these data directly without doing any memory copy. When the frame locates in multiple BD buffer, need to repeat calling this function until *isLastBuff*=true (need to store the temp buf pointer everytime call this function). After finishing the analysis of this frame, call `ENET_ReleaseRxBuffer` to release rxbuff memory to DMA. This is an example:

```
*      uint32_t length;
*      uint8_t *buf = NULL;
*      uint32_t data_len = 0;
*      bool isLastBuff = false;
*      enet_handle_t g_handle;
*      status_t status;
*      status = ENET_GetRxFrameSize(&g_handle, &length, 0);
*      if (length != 0)
*      {
*          ENET_GetRxBuffer(EXAMPLE_ENET, &g_handle, &buf, &data_len, 0, &isLastBuff, NULL
*      );
*          ENET_ReleaseRxBuffer(EXAMPLE_ENET, &g_handle, buf, 0);
*      }
*
```

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>buffer</i>	The data buffer pointer to store the frame.
<i>length</i>	The size of the data buffer. If <i>isLastBuff</i> =false, it represents data length of this buffer. If <i>isLastBuff</i> =true, it represents data length of total frame.

<i>ringId</i>	The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEEn(x) - 1).
<i>isLastBuff</i>	The flag represents whether this buffer is the last buffer to store frame.
<i>ts</i>	The 1588 timestamp value, valid in last buffer.

Return values

<i>kStatus_Success</i>	Get receive buffer succeed.
<i>kStatus_ENET_RxFrame-Fail</i>	Get receive buffer fails, it's owned by application, should wait app to release this buffer.

28.8.41 void ENET_ReleaseRxBuffer (ENET_Type * *base*, enet_handle_t * *handle*, void * *buffer*, uint8_t *ringId*)

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

This function can release specified BD owned by application, meanwhile it may rearrange the BD to let the no-owned BDs always in back of the index of DMA transfer. So for the situation that releasing order is not same as the getting order, the rearrangement makes all ready BDs can be used by DMA.

Note

This function can't be interrupted by ENET_GetRxBuffer, so in application must make sure ENET_GetRxBuffer is called before or after this function. And this function itself isn't thread safe due to BD content exchanging.

Parameters

<i>base</i>	ENET peripheral base address.
<i>handle</i>	The ENET handler structure. This is the same handler pointer used in the ENET_Init.
<i>buffer</i>	The buffer address to store frame, using it to find the correspond BD and release it.
<i>ringId</i>	The ring index, range from 0 ~ (FSL_FEATURE_ENET_INSTANCE_QUEUEEn(x) - 1).

28.8.42 status_t ENET_GetRxFrame (ENET_Type * *base*, enet_handle_t * *handle*, enet_rx_frame_struct_t * *rxFrame*, uint8_t *ringId*)

This function will use the user-defined allocate and free callback. Every time application gets one frame through this function, driver will allocate new buffers for the BDs whose buffers have been taken by application.

Note

This function will drop current frame and update related BDs as available for DMA if new buffers allocating fails. Application must provide a memory pool including at least BD number + 1 buffers to make this function work normally. If user calls this function in Rx interrupt handler, be careful that this function makes Rx BD ready with allocating new buffer(normal) or updating current BD(out of memory). If there's always new Rx frame input, Rx interrupt will be triggered forever. Application need to disable Rx interrupt according to specific design in this case.

Parameters

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

Return values

<i>kStatus_Success</i>	Succeed to get one frame and allocate new memory for Rx buffer.
<i>kStatus_ENET_RxFrame-Empty</i>	There's no Rx frame in the BD.
<i>kStatus_ENET_RxFrame-Error</i>	There's issue in this receiving.
<i>kStatus_ENET_RxFrame-Drop</i>	There's no new buffer memory for BD, drop this frame.

28.8.43 **status_t ENET_StartTxFrame (ENET_Type * *base*, enet_handle_t * *handle*, enet_tx_frame_struct_t * *txFrame*, uint8_t *ringId*)**

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

Note

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

Parameters

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

Return values

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

28.8.44 `status_t ENET_SendFrameZeroCopy (ENET_Type * base, enet_handle_t * handle, const uint8_t * data, uint32_t length, uint8_t ringId, bool tsFlag, void * context)`

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

Note

The CRC is automatically appended to the data. Input the data to send without the CRC. The frame must store in continuous memory and need to check the buffer start address alignment based on your device, otherwise it has issue or can't get highest DMA transmit speed.

Parameters

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

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

Return values

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

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

Parameters

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

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

Parameters

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

28.8.47 void ENET_CommonFrame1IRQHandler (ENET_Type * *base*)

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

Parameters

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

28.8.48 void ENET_CommonFrame2IRQHandler (ENET_Type * *base*)

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

Parameters

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

28.8.49 void ENET_ErrorIRQHandler (ENET_Type * *base*, enet_handle_t * *handle*)

Parameters

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

28.8.50 void ENET_Ptp1588IRQHandler (ENET_Type * *base*)

This is used for the 1588 timer interrupt.

Parameters

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

28.8.51 void ENET_CommonFrame0IRQHandler (ENET_Type * *base*)

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

Parameters

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

28.9 Variable Documentation

28.9.1 const clock_ip_name_t s_enetClock[]

28.10 ENET CMSIS Driver

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

The ENET CMSIS driver includes transactional APIs.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code accessing the hardware registers.

28.10.1 Typical use case

```
void ENET_SignalEvent_t(uint32_t event)
{
    if (event == ARM_ETH_MAC_EVENT_RX_FRAME)
    {
        uint32_t size;
        uint32_t len;

        /* Get the Frame size */
        size = EXAMPLE_ENET.GetRxFrameSize();
        /* Call ENET_ReadFrame when there is a received frame. */
        if (size != 0)
        {
            /* Received valid frame. Deliver the rx buffer with the size equal to length. */
            uint8_t *data = (uint8_t *)malloc(size);
            if (data)
            {
                len = EXAMPLE_ENET.ReadFrame(data, size);
                if (size == len)
                {
                    /* Increase the received frame numbers. */
                    if (g_rxIndex < ENET_EXAMPLE_LOOP_COUNT)
                    {
                        g_rxIndex++;
                    }
                }
                free(data);
            }
        }
    }
    if (event == ARM_ETH_MAC_EVENT_TX_FRAME)
    {
        g_testTxNum ++;
    }
}

/* Initialize the ENET module. */
EXAMPLE_ENET.Initialize(ENET_SignalEvent_t);
EXAMPLE_ENET.PowerControl(ARM_POWER_FULL);
EXAMPLE_ENET.SetMacAddress((ARM_ETH_MAC_ADDR *)g_macAddr);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONFIGURE, linkInfo.speed << ARM_ETH_MAC_SPEED_Pos |
    linkInfo.duplex << ARM_ETH_MAC_DUPLEX_Pos | ARM_ETH_MAC_ADDRESS_BROADCAST);
EXAMPLE_ENET_PHY.PowerControl(ARM_POWER_FULL);
```

```

EXAMPLE_ENET_PHY.SetMode(ARM_ETH_PHY_AUTO_NEGOTIATE);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_RX, 1);
EXAMPLE_ENET.Control(ARM_ETH_MAC_CONTROL_TX, 1);
if (EXAMPLE_ENET_PHY.GetLinkState() == ARM_ETH_LINK_UP)
{
    linkInfo = EXAMPLE_ENET_PHY.GetLinkInfo();
}
else
{
    PRINTF("\r\nPHY Link down, please check the cable connection and link partner setting.\r\n");
}

/* Build broadcast for sending. */
ENET_BuildBroadCastFrame();

while (1)
{
    /* Check the total number of received number. */
    if (g_rxCheckIdx != g_rxIndex)
    {
        PRINTF("The %d frame has been successfully received!\r\n", g_rxIndex);
        g_rxCheckIdx = g_rxIndex;
    }
    if (g_testTxNum && (g_txCheckIdx != g_testTxNum))
    {
        g_txCheckIdx = g_testTxNum;
        PRINTF("The %d frame transmitted success!\r\n", g_txCheckIdx);
    }
    /* Get the Frame size */
    if (txnumber < ENET_EXAMPLE_LOOP_COUNT)
    {
        txnumber++;
        /* Send a multicast frame when the PHY is link up. */
        if (EXAMPLE_ENET.SendFrame(&g_frame[0], ENET_DATA_LENGTH, ARM_ETH_MAC_TX_FRAME_EVENT) ==
ARM_DRIVER_OK)
        {
            for (uint32_t count = 0; count < 0x3FF; count++)
            {
                __ASM("nop");
            }
        }
        else
        {
            PRINTF("\r\nTransmit frame failed!\r\n");
        }
    }
}

```


Chapter 29

EQOS-TSN: Ethernet QoS with TSN Driver

The MCUXpresso SDK provides a peripheral driver for the 10/100/1000 Mbps Ethernet QoS with TSN module of MCUXpresso SDK devices.

29.1 EQOS-TSN: Ethernet QoS with TSN Driver

Operations of Ethernet QoS with TSN Driver {EthernetQosDriverOps}

29.1.1 Initialize and De-initialize interface Operation

Use the [ENET_QOS_GetDefaultConfig\(\)](#) to get the default basic configuration, Use the default configuration unchanged or changed as the input to the [ENET_QOS_Init\(\)](#) to do basic configuration for EQOS module. Call [ENET_QOS_DescriptorInit\(\)](#) to initialization the descriptors and Call [ENET_QOS_StartRxTx\(\)](#) to start the EQOS engine after all initialization. [ENET_QOS_Deinit\(\)](#) is used to EQOS De-initialization.

29.1.2 Other basic operation

This group provides the EQOS mac address set/get operation with [ENET_QOS_SetMacAddr\(\)](#) and [ENET_QOS_GetMacAddr\(\)](#). The [ENET_QOS_EnterPowerDown\(\)](#) and [ENET_QOS_ExitPowerDown\(\)](#) can be used to do power management.

29.1.3 Interrupt operation

This group provide the DMA interrupt get and clear APIs. This can be used by application to create new IRQ handler.

29.1.4 Functional Operation

This group functions are low level TX/RX descriptor operations. It is convenient to use these TX/RX APIs to do application specific RX/TX. For TX: Use [ENET_QOS_IsTxDescriptorDmaOwn\(\)](#), [ENET_QOS_SetupTxDescriptor\(\)](#) to build your packet for transfer and [ENET_QOS_UpdateTxDescriptorTail\(\)](#) to update the TX tail pointer. For RX: Use [ENET_QOS_GetRxDescriptor\(\)](#) to get the received data/length and use the [ENET_QOS_UpdateRxDescriptor\(\)](#) to update the buffers/status.

29.1.5 Transactional Operation

When use the Transactional APIs, please make sure to call the `ENET_QOS_CreateHandler` to create the handler which are used to maintain all data related to TX/RX process.

For EQOS receive, the `ENET_QOS_GetRxFrameSize()` function must be called to get the received data size. Then, call the `ENET_QOS_ReadFrame()` function to get the received data.

For EQOS transmit, call the `ENET_QOS_SendFrame()` function to send the data out. To save memory and avoid the memory copy in the TX process. The `ENET_QOS_SendFrame()` here is a zero-copy API, so make sure the input data buffers are not re-queued or freed before the data are really sent out. To make sure the data buffers reclaim is rightly done. the transmit interrupt must be used. so For transactional APIs here we enabled the tx interrupt in `ENET_QOS_CreateHandler()`. That means the tx interrupt is automatically enabled in transactional APIs. is recommended to be called on the transmit interrupt handler. `ENET_QOS_ReclaimTxDescriptor()` is a transactional API to get the information from the finished transmit data buffers and reclaim the tx index. it is called by the transmit interrupt IRQ handler.

For use the transactional APIs, receive polling Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet_qos` For the functional API, rx polling Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/enet_qos`

Chapter 30

EWM: External Watchdog Monitor Driver

30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the External Watchdog (EWM) Driver module of MCUXpresso SDK devices.

30.2 Typical use case

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

Data Structures

- struct `ewm_config_t`
Data structure for EWM configuration. [More...](#)

Enumerations

- enum `ewm_lpo_clock_source_t` {
 `kEWM_LpoClockSource0` = 0U,
 `kEWM_LpoClockSource1` = 1U,
 `kEWM_LpoClockSource2` = 2U,
 `kEWM_LpoClockSource3` = 3U }
Describes EWM clock source.
- enum `_ewm_interrupt_enable_t` { `kEWM_InterruptEnable` = EWM_CTRL_INTEN_MASK }
EWM interrupt configuration structure with default settings all disabled.
- enum `_ewm_status_flags_t` { `kEWM_RunningFlag` = EWM_CTRL_EWMEN_MASK }
EWM status flags.

Driver version

- #define `FSL_EWM_DRIVER_VERSION` (MAKE_VERSION(2, 0, 3))
EWM driver version 2.0.3.

EWM initialization and de-initialization

- void `EWM_Init` (EWM_Type *base, const `ewm_config_t` *config)
Initializes the EWM peripheral.
- void `EWM_Deinit` (EWM_Type *base)
Deinitializes the EWM peripheral.
- void `EWM_GetDefaultConfig` (`ewm_config_t` *config)
Initializes the EWM configuration structure.

EWM functional Operation

- static void [EWM_EnableInterrupts](#) (EWM_Type *base, uint32_t mask)
Enables the EWM interrupt.
- static void [EWM_DisableInterrupts](#) (EWM_Type *base, uint32_t mask)
Disables the EWM interrupt.
- static uint32_t [EWM_GetStatusFlags](#) (EWM_Type *base)
Gets all status flags.
- void [EWM_Refresh](#) (EWM_Type *base)
Serves the EWM.

30.3 Data Structure Documentation

30.3.1 struct ewm_config_t

This structure is used to configure the EWM.

Data Fields

- bool [enableEwm](#)
Enable EWM module.
- bool [enableEwmInput](#)
Enable EWM_in input.
- bool [setInputAssertLogic](#)
EWM_in signal assertion state.
- bool [enableInterrupt](#)
Enable EWM interrupt.
- [ewm_lpo_clock_source_t](#) clockSource
Clock source select.
- uint8_t [prescaler](#)
Clock prescaler value.
- uint8_t [compareLowValue](#)
Compare low-register value.
- uint8_t [compareHighValue](#)
Compare high-register value.

30.4 Macro Definition Documentation

30.4.1 #define FSL_EWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

30.5 Enumeration Type Documentation

30.5.1 enum ewm_lpo_clock_source_t

Enumerator

- kEWM_LpoClockSource0* EWM clock sourced from lpo_clk[0].
- kEWM_LpoClockSource1* EWM clock sourced from lpo_clk[1].
- kEWM_LpoClockSource2* EWM clock sourced from lpo_clk[2].

kEWM_LpoClockSource3 EWM clock sourced from lpo_clk[3].

30.5.2 enum _ewm_interrupt_enable_t

This structure contains the settings for all of EWM interrupt configurations.

Enumerator

kEWM_InterruptEnable Enable the EWM to generate an interrupt.

30.5.3 enum _ewm_status_flags_t

This structure contains the constants for the EWM status flags for use in the EWM functions.

Enumerator

kEWM_RunningFlag Running flag, set when EWM is enabled.

30.6 Function Documentation

30.6.1 void EWM_Init (EWM_Type * *base*, const ewm_config_t * *config*)

This function is used to initialize the EWM. After calling, the EWM runs immediately according to the configuration. Note that, except for the interrupt enable control bit, other control bits and registers are write once after a CPU reset. Modifying them more than once generates a bus transfer error.

This is an example.

```
*  ewm_config_t config;
*  EWM_GetDefaultConfig(&config);
*  config.compareHighValue = 0xAAU;
*  EWM_Init(ewm_base, &config);
*
```

Parameters

<i>base</i>	EWM peripheral base address
<i>config</i>	The configuration of the EWM

30.6.2 void EWM_Deinit (EWM_Type * *base*)

This function is used to shut down the EWM.

Parameters

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

30.6.3 void EWM_GetDefaultConfig (ewm_config_t * *config*)

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

```
* ewmConfig->enableEwm = true;
* ewmConfig->enableEwmInput = false;
* ewmConfig->setInputAssertLogic = false;
* ewmConfig->enableInterrupt = false;
* ewmConfig->ewm_lpo_clock_source_t = kEWM_LpoClockSource0;
* ewmConfig->prescaler = 0;
* ewmConfig->compareLowValue = 0;
* ewmConfig->compareHighValue = 0xFEU;
*
```

Parameters

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

See Also

[ewm_config_t](#)

**30.6.4 static void EWM_EnableInterrupts (EWM_Type * *base*, uint32_t *mask*)
[inline], [static]**

This function enables the EWM interrupt.

Parameters

<i>base</i>	EWM peripheral base address
<i>mask</i>	The interrupts to enable The parameter can be combination of the following source if defined <ul style="list-style-type: none"> • kEWM_InterruptEnable

**30.6.5 static void EWM_DisableInterrupts (EWM_Type * *base*, uint32_t *mask*)
[inline], [static]**

This function enables the EWM interrupt.

Parameters

<i>base</i>	EWM peripheral base address
<i>mask</i>	The interrupts to disable The parameter can be combination of the following source if defined <ul style="list-style-type: none"> • kEWM_InterruptEnable

30.6.6 static uint32_t EWM_GetStatusFlags (EWM_Type * *base*) [inline], [static]

This function gets all status flags.

This is an example for getting the running flag.

```
*  uint32_t status;
*  status = EWM_GetStatusFlags(ewm_base) & kEWM_RunningFlag;
*
```

Parameters

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

Returns

State of the status flag: asserted (true) or not-asserted (false).

See Also

[_ewm_status_flags_t](#)

- True: a related status flag has been set.
- False: a related status flag is not set.

30.6.7 void EWM_Refresh (EWM_Type * *base*)

This function resets the EWM counter to zero.



Parameters

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



Chapter 31

FlexCAN: Flex Controller Area Network Driver

31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Flex Controller Area Network (FlexCAN) module of MCUXpresso SDK devices.

Modules

- [FlexCAN Driver](#)
- [FlexCAN eDMA Driver](#)

31.2 FlexCAN Driver

31.2.1 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.

31.2.2 Typical use case

31.2.2.1 Message Buffer Send Operation

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

31.2.2.2 Message Buffer Receive Operation

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

31.2.2.3 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_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 `FLEXCAN_MECR_INT_MASK`(x) (((uint64_t)((uint64_t)(x)) << 16)) & 0xD00000000U)
FlexCAN interrupt/status flag 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_ErrorStatus = MAKE_STATUS(kStatusGroup_FLEXCAN, 10),
 kStatus_FLEXCAN_WakeUp = MAKE_STATUS(kStatusGroup_FLEXCAN, 11),
 kStatus_FLEXCAN_UnHandled = MAKE_STATUS(kStatusGroup_FLEXCAN, 12),
 kStatus_FLEXCAN_RxRemote = MAKE_STATUS(kStatusGroup_FLEXCAN, 13) }
 FlexCAN Enhanced Rx FIFO base address helper macro.
- 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_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_HostAccessNErrorInterruptEnable = FLEXCAN_MECR_INT_MASK(CAN_MECR_HANCEI_MSK_MASK),
kFLEXCAN_FlexCanAccessNErrorInterruptEnable = FLEXCAN_MECR_INT_MASK(CAN_MECR_FANCEI_MSK_MASK),
kFLEXCAN_HostOrFlexCanCErrorInterruptEnable = FLEXCAN_MECR_INT_MASK(CAN_MECR_CEI_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_HostAccessNonCorrectableErrorIntFlag = FLEXCAN_MECR_INT_MASK(CAN_
ERRSR_HANCEIF_MASK),
kFLEXCAN_FlexCanAccessNonCorrectableErrorIntFlag = FLEXCAN_MECR_INT_MASK(CA
N_ERRSR_FANCEIF_MASK),
kFLEXCAN_CorrectableErrorIntFlag = FLEXCAN_MECR_INT_MASK(CAN_ERRSR_CEIF_
MASK),
kFLEXCAN_HostAccessNonCorrectableErrorOverrunFlag = FLEXCAN_MECR_INT_MASK(C
AN_ERRSR_HANCEIOF_MASK),
kFLEXCAN_FlexCanAccessNonCorrectableErrorOverrunFlag = FLEXCAN_MECR_INT_MAS
K(CAN_ERRSR_FANCEIOF_MASK),
kFLEXCAN_CorrectableErrorOverrunFlag = FLEXCAN_MECR_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 Error 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, 8, 2))`
FlexCAN driver version.

Initialization and deinitialization

- void `FLEXCAN_EnterFreezeMode` (CAN_Type *base)
Enter FlexCAN Freeze Mode.
- void `FLEXCAN_ExitFreezeMode` (CAN_Type *base)
Exit FlexCAN Freeze Mode.
- uint32_t `FLEXCAN_GetInstance` (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_FDInit** (CAN_Type *base, const flexcan_config_t *pConfig, uint32_t sourceClock_Hz, flexcan_mb_size_t dataSize, bool brs)
Initializes a FlexCAN instance.
- 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 protocol timing characteristic.
- void **FLEXCAN_SetFDTimingConfig** (CAN_Type *base, const flexcan_timing_config_t *pConfig)
Sets the FlexCAN CANFD data phase timing characteristic.
- void **FLEXCAN_SetRxMbGlobalMask** (CAN_Type *base, uint32_t mask)
Sets the FlexCAN receive message buffer global mask.
- void **FLEXCAN_SetRxFifoGlobalMask** (CAN_Type *base, uint32_t mask)
Sets the FlexCAN receive FIFO global mask.
- void **FLEXCAN_SetRxIndividualMask** (CAN_Type *base, uint8_t maskIdx, uint32_t mask)
Sets the FlexCAN receive individual mask.
- void **FLEXCAN_SetTxMbConfig** (CAN_Type *base, uint8_t mbIdx, bool enable)
Configures a FlexCAN transmit message buffer.
- void **FLEXCAN_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.

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)
Gets the FlexCAN Bus Error Counter value.
- static uint64_t [FLEXCAN_GetMbStatusFlags](#) (CAN_Type *base, uint64_t mask)
Gets the FlexCAN Message Buffer interrupt flags.
- static void [FLEXCAN_ClearMbStatusFlags](#) (CAN_Type *base, uint64_t mask)
Clears the FlexCAN Message Buffer interrupt flags.
- void [FLEXCAN_GetMemoryErrorReportStatus](#) (CAN_Type *base, [flexcan_memory_error_report_status_t](#) *errorStatus)
Gets the FlexCAN Memory Error Report registers status.

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_DisableMbInterrupts](#) (CAN_Type *base, uint64_t mask)
Disables FlexCAN Message Buffer interrupts.

DMA Control

- void [FLEXCAN_EnableRxFifoDMA](#) (CAN_Type *base, bool enable)
Enables or disables the FlexCAN Rx FIFO DMA request.
- static uint32_t [FLEXCAN_GetRxFifoHeadAddr](#) (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.

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.
- void [FLEXCAN_TransferCreateHandle](#) (CAN_Type *base, flexcan_handle_t *handle, flexcan_transfer_callback_t callback, void *userData)
Initializes the FlexCAN handle.
- [status_t FLEXCAN_TransferSendNonBlocking](#) (CAN_Type *base, flexcan_handle_t *handle, [flexcan_mb_transfer_t](#) *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.
- 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 mb-

Idx)

Aborts the interrupt driven message receive process.

- void [FLEXCAN_TransferAbortReceiveFifo](#) (CAN_Type *base, flexcan_handle_t *handle)

Aborts the interrupt driven message receive from Rx FIFO process.

- void [FLEXCAN_TransferHandleIRQ](#) (CAN_Type *base, flexcan_handle_t *handle)

FlexCAN IRQ handle function.

31.2.3 Data Structure Documentation

31.2.3.1 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.

31.2.3.2 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`

31.2.3.3 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](#).

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`

31.2.3.4 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

31.2.3.5 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 [disableSelfReception](#)

- *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.*
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`

31.2.3.6 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 of 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`

31.2.3.7 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`

31.2.3.8 struct `flexcan_mb_transfer_t`

Data Fields

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

Field Documentation

(1) `flexcan_frame_t* flexcan_mb_transfer_t::frame`

(2) `uint8_t flexcan_mb_transfer_t::mbIdx`

31.2.3.9 struct `flexcan_fifo_transfer_t`

Data Fields

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

Field Documentation

(1) `flexcan_frame_t* flexcan_fifo_transfer_t::frame`

31.2.3.10 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.
- `volatile uint8_t mbState [CAN_WORD1_COUNT]`
Message Buffer transfer state.
- `volatile uint8_t rxFifoState`
Rx FIFO transfer state.
- `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) `volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]`
- (7) `volatile uint8_t flexcan_handle_t::rxFifoState`
- (8) `volatile uint32_t flexcan_handle_t::timestamp[CAN_WORD1_COUNT]`

31.2.4 Macro Definition Documentation

31.2.4.1 `#define FSL_FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 8, 2))`

31.2.4.2 `#define DLC_LENGTH_DECODE(dlc) (((dlc) <= 8U) ? (dlc) : (((dlc) <= 12U) ? (((dlc)-6U) * 4U) : (((dlc)-11U) * 16U)))`

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

Standard Frame ID helper macro.

31.2.4.4 `#define FLEXCAN_ID_EXT(id)`

Value:

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

31.2.4.5 `#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.

31.2.4.6 #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))
```

31.2.4.7 #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.

31.2.4.8 #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))
```

31.2.4.9 #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))
```

**31.2.4.10 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(*id*
) (((uint32_t)(id)&0x7F8) << 21)****31.2.4.11 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(*id*
) (((uint32_t)(id)&0x7F8) << 13)****31.2.4.12 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(*id*
) (((uint32_t)(id)&0x7F8) << 5)****31.2.4.13 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(*id*
) (((uint32_t)(id)&0x7F8) >> 3)****31.2.4.14 #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))
```

31.2.4.15 #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))
```

31.2.4.16 #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))
```

31.2.4.17 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(*id*) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)

31.2.4.18 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(*id*)

Value:

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

31.2.4.19 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(*id*)

Value:

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

31.2.4.20 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(*id*) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)

31.2.4.21 #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.

31.2.4.22 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(*id*, *rtr*, *ide*)**Value:**

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

31.2.4.23 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(*id*, *rtr*, *ide*)**Value:**

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

31.2.4.24 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(*id*)**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(         \
    id)
```

31.2.4.25 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(*id*)**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(     \
    id)
```

31.2.4.26 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(*id*)**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(      \
    id)
```

31.2.4.27 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(*id*)**Value:**

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(          \
    id)
```

**31.2.4.28 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(*id*, *rtr*, *ide*
) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)**

31.2.4.29 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(*id*, *rtr*, *ide*)

Value:

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

31.2.4.30 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(*id*, *rtr*, *ide*)

Value:

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

31.2.4.31 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(          \
    id)
```

31.2.4.32 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(      \
    id)
```

31.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(       \
    id)
```

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

31.2.4.35 `#define FLEXCAN_MECR_INT_MASK(x) (((uint64_t)(((uint64_t)(x)) << 16)) & 0xD0000000U)`

31.2.4.36 `#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.

31.2.5 Enumeration Type Documentation

31.2.5.1 anonymous enum

FlexCAN transfer status.

Enumerator

kStatus_FLEXCAN_TxBusy Tx Message Buffer is Busy.
kStatus_FLEXCAN_TxIdle Tx Message Buffer is Idle.
kStatus_FLEXCAN_TxSwitchToRx Remote Message is send out and Message buffer changed to Receive one.
kStatus_FLEXCAN_RxBusy Rx Message Buffer is Busy.
kStatus_FLEXCAN_RxIdle Rx Message Buffer is Idle.
kStatus_FLEXCAN_RxOverflow Rx Message Buffer is Overflowed.
kStatus_FLEXCAN_RxFifoBusy Rx Message FIFO is Busy.
kStatus_FLEXCAN_RxFifoIdle Rx Message FIFO is Idle.
kStatus_FLEXCAN_RxFifoOverflow Rx Message FIFO is overflowed.
kStatus_FLEXCAN_RxFifoWarning Rx Message FIFO is almost overflowed.
kStatus_FLEXCAN_ErrorStatus FlexCAN Module Error and Status.
kStatus_FLEXCAN_WakeUp FlexCAN is waken up from STOP mode.
kStatus_FLEXCAN_UnHandled UnHandled Interrupt asserted.
kStatus_FLEXCAN_RxRemote Rx Remote Message Received in Mail box.

31.2.5.2 enum flexcan_frame_format_t

Enumerator

kFLEXCAN_FrameFormatStandard Standard frame format attribute.

kFLEXCAN_FrameFormatExtend Extend frame format attribute.

31.2.5.3 enum flexcan_frame_type_t

Enumerator

kFLEXCAN_FrameTypeData Data frame type attribute.

kFLEXCAN_FrameTypeRemote Remote frame type attribute.

31.2.5.4 enum flexcan_clock_source_t

Deprecated Do not use the ***kFLEXCAN_ClkSrcOs***. It has been superseded ***kFLEXCAN_ClkSrc0***
Do not use the ***kFLEXCAN_ClkSrcPeri***. It has been superseded ***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.

31.2.5.5 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.

31.2.5.6 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.

31.2.5.7 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.

31.2.5.8 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.
- kFLEXCAN_1BperFrame*** Frame contains 1 valid data bytes.
- kFLEXCAN_2BperFrame*** Frame contains 2 valid data bytes.
- kFLEXCAN_3BperFrame*** Frame contains 3 valid data bytes.
- kFLEXCAN_4BperFrame*** Frame contains 4 valid data bytes.
- kFLEXCAN_5BperFrame*** Frame contains 5 valid data bytes.
- kFLEXCAN_6BperFrame*** Frame contains 6 valid data bytes.
- kFLEXCAN_7BperFrame*** Frame contains 7 valid data bytes.
- kFLEXCAN_8BperFrame*** Frame contains 8 valid data bytes.
- kFLEXCAN_12BperFrame*** Frame contains 12 valid data bytes.
- kFLEXCAN_16BperFrame*** Frame contains 16 valid data bytes.
- kFLEXCAN_20BperFrame*** Frame contains 20 valid data bytes.
- kFLEXCAN_24BperFrame*** Frame contains 24 valid data bytes.
- kFLEXCAN_32BperFrame*** Frame contains 32 valid data bytes.
- kFLEXCAN_48BperFrame*** Frame contains 48 valid data bytes.
- kFLEXCAN_64BperFrame*** Frame contains 64 valid data bytes.

31.2.5.9 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.

31.2.5.10 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 22.
kFLEXCAN_FDErrorInterruptEnable CAN FD Error interrupt, use bit 31.
kFLEXCAN_HostAccessNCErrortInterruptEnable Host Access With Non-Correctable Errors interrupt, use high word bit 0.
kFLEXCAN_FlexCanAccessNCErrortInterruptEnable FlexCAN Access With Non-Correctable Errors interrupt, use high word bit 2.
kFLEXCAN_HostOrFlexCanCErrortInterruptEnable Host or FlexCAN Access With Correctable Errors interrupt, use high word bit 3.

31.2.5.11 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_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.

31.2.5.12 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.

31.2.5.13 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.

31.2.5.14 enum flexcan_memory_error_type_t

Enumerator

kFLEXCAN_CorrectableError The memory error is correctable which means on bit error.
kFLEXCAN_NonCorrectableError The memory error is non-correctable which means two bit errors.

31.2.5.15 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.

31.2.5.16 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.

31.2.6 Function Documentation

31.2.6.1 void FLEXCAN_EnterFreezeMode (CAN_Type * *base*)

This function makes the FlexCAN work under Freeze Mode.

Parameters

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

31.2.6.2 void FLEXCAN_ExitFreezeMode (CAN_Type * *base*)

This function makes the FlexCAN leave Freeze Mode.

Parameters

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

31.2.6.3 uint32_t FLEXCAN_GetInstance (CAN_Type * *base*)

Parameters

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

Returns

FlexCAN instance.

31.2.6.4 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.

31.2.6.5 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.

31.2.6.6 **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

31.2.6.7 **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(CAN0, &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.

31.2.6.8 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.
-------------	----------------------------------

31.2.6.9 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->enableNonCorrectableErrorEnterFreeze = true; flexcanConfig.timingConfig = timingConfig;

Parameters

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

31.2.6.10 void FLEXCAN_SetTimingConfig (CAN_Type * *base*, const flexcan_timing_config_t * *pConfig*)

This function gives user settings to classical CAN or CANFD nominal phase timing characteristic. The function is for an experienced user. For less experienced users, call the [FLEXCAN_GetDefaultConfig\(\)](#) and get the default timing characteristic, then call [FLEXCAN_Init\(\)](#) and fill the bit rate field.

Note

Calling [FLEXCAN_SetTimingConfig\(\)](#) overrides the bit rate set in [FLEXCAN_Init\(\)](#).

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pConfig</i>	Pointer to the timing configuration structure.

31.2.6.11 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_GetDefaultConfig\(\)](#) and get the default timing characteristic, then call [FLEXCAN_FDInit\(\)](#) and fill the data phase bit rate field.

Note

Calling [FLEXCAN_SetFDTimingConfig\(\)](#) overrides the bit rate set in [FLEXCAN_FDInit\(\)](#).

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pConfig</i>	Pointer to the timing configuration structure.

31.2.6.12 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.

31.2.6.13 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.

31.2.6.14 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.

31.2.6.15 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"> • true: Enable Tx Message Buffer. • false: Disable Tx Message Buffer.

31.2.6.16 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"> • true: Enable Tx Message Buffer. • false: Disable Tx Message Buffer.

31.2.6.17 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"> • true: Enable Rx Message Buffer. • false: Disable Rx Message Buffer.

31.2.6.18 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"> • true: Enable Rx Message Buffer. • false: Disable Rx Message Buffer.

31.2.6.19 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"> • true: Enable Legacy Rx FIFO. • false: Disable Legacy Rx FIFO.

31.2.6.20 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](#).

31.2.6.21 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 _flexcan_flags .

31.2.6.22 `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.

31.2.6.23 `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.

31.2.6.24 `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.

31.2.6.25 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.

31.2.6.26 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 _flexcan_interrupt_enable .

31.2.6.27 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 _flexcan_interrupt_enable .

31.2.6.28 `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.

31.2.6.29 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.

31.2.6.30 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.

31.2.6.31 static uint32_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.

31.2.6.32 `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.

31.2.6.33 **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.

31.2.6.34 **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.

31.2.6.35 **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.

31.2.6.36 **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.

31.2.6.37 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.

31.2.6.38 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.

31.2.6.39 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.

31.2.6.40 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 flexcan_mb_transfer_t .

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.

31.2.6.41 `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 flexcan_mb_transfer_t .

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.

31.2.6.42 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.

31.2.6.43 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.

31.2.6.44 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.

31.2.6.45 **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.

31.2.6.46 **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.

31.2.6.47 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.

31.2.6.48 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 flexcan_mb_transfer_t .

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.

31.2.6.49 **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 flexcan_mb_transfer_t .

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.

31.2.6.50 **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 flexcan_fifo_transfer_t .

Return values

<i>kStatus_Success</i>	- Start Rx FIFO receiving process successfully.
<i>kStatus_FLEXCAN_Rx-FifoBusy</i>	- Rx FIFO is currently in use.

31.2.6.51 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.
------------	---

31.2.6.52 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.

31.2.6.53 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.

31.2.6.54 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.

31.2.6.55 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.

31.3 FlexCAN eDMA Driver

31.3.1 Overview

Data Structures

- struct [flexcan_edma_handle_t](#)
FlexCAN eDMA handle. [More...](#)

Typedefs

- typedef void(* [flexcan_edma_transfer_callback_t](#))(CAN_Type *base, flexcan_edma_handle_t *handle, [status_t](#) status, void *userData)
FlexCAN transfer callback function.

Driver version

- #define [FSL_FLEXCAN_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 8, 1))
FlexCAN EDMA driver version.

eDMA transactional

- void [FLEXCAN_TransferCreateHandleEDMA](#) (CAN_Type *base, flexcan_edma_handle_t *handle, [flexcan_edma_transfer_callback_t](#) callback, void *userData, [edma_handle_t](#) *rxFifoEdmaHandle)
Initializes the FlexCAN handle, which is used in transactional functions.
- void [FLEXCAN_PrepareTransfConfiguration](#) (CAN_Type *base, [flexcan_fifo_transfer_t](#) *pFifoXfer, [edma_transfer_config_t](#) *pEdmaConfig)
Prepares the eDMA transfer configuration for FLEXCAN Legacy RX FIFO.
- [status_t](#) [FLEXCAN_StartTransferDatafromRxFIFO](#) (CAN_Type *base, flexcan_edma_handle_t *handle, [edma_transfer_config_t](#) *pEdmaConfig)
Start Transfer Data from the FLEXCAN Legacy Rx FIFO using eDMA.
- [status_t](#) [FLEXCAN_TransferReceiveFifoEDMA](#) (CAN_Type *base, flexcan_edma_handle_t *handle, [flexcan_fifo_transfer_t](#) *pFifoXfer)
Receives the CAN Message from the Legacy Rx FIFO using eDMA.
- void [FLEXCAN_TransferAbortReceiveFifoEDMA](#) (CAN_Type *base, flexcan_edma_handle_t *handle)
Aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

31.3.2 Data Structure Documentation

31.3.2.1 struct _flexcan_edma_handle

Data Fields

- [flexcan_edma_transfer_callback_t](#) *callback*
Callback function.
- void * [userData](#)
FlexCAN callback function parameter.
- [edma_handle_t](#) * [rxFifoEdmaHandle](#)
The EDMA handler for Rx FIFO.
- volatile uint8_t [rxFifoState](#)
Rx FIFO transfer state.

Field Documentation

- (1) [flexcan_edma_transfer_callback_t](#) [flexcan_edma_handle_t::callback](#)
- (2) void* [flexcan_edma_handle_t::userData](#)
- (3) [edma_handle_t](#)* [flexcan_edma_handle_t::rxFifoEdmaHandle](#)
- (4) volatile uint8_t [flexcan_edma_handle_t::rxFifoState](#)

31.3.3 Macro Definition Documentation

31.3.3.1 #define FSL_FLEXCAN_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 8, 1))

31.3.4 Typedef Documentation

31.3.4.1 typedef void(* flexcan_edma_transfer_callback_t)(CAN_Type *base, flexcan_edma_handle_t *handle, status_t status, void *userData)

31.3.5 Function Documentation

31.3.5.1 void FLEXCAN_TransferCreateHandleEDMA (CAN_Type * base, flexcan_edma_handle_t * handle, flexcan_edma_transfer_callback_t callback, void * userData, edma_handle_t * rxFifoEdmaHandle)

Parameters

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

<i>handle</i>	Pointer to flexcan_edma_handle_t structure.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.
<i>rxFifoEdma-Handle</i>	User-requested DMA handle for Rx FIFO DMA transfer.

31.3.5.2 void FLEXCAN_PrepareTransfConfiguration (CAN_Type * *base*, flexcan_fifo_transfer_t * *pFifoXfer*, edma_transfer_config_t * *pEdmaConfig*)

This function prepares the eDMA transfer configuration structure according to FLEXCAN Legacy RX FIFO.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>pFifoXfer</i>	FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t .
<i>pEdmaConfig</i>	The user configuration structure of type edma_transfer_t.

31.3.5.3 status_t FLEXCAN_StartTransferDatafromRxFIFO (CAN_Type * *base*, flexcan_edma_handle_t * *handle*, edma_transfer_config_t * *pEdmaConfig*)

This function to Update edma transfer configuration and Start eDMA transfer

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	Pointer to flexcan_edma_handle_t structure.
<i>pEdmaConfig</i>	The user configuration structure of type edma_transfer_t.

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_FLEXCAN_Rx-FifoBusy</i>	Previous transfer ongoing.

31.3.5.4 `status_t FLEXCAN_TransferReceiveFifoEDMA (CAN_Type * base,
flexcan_edma_handle_t * handle, flexcan_fifo_transfer_t * pFifoXfer)`

This function receives the CAN Message using eDMA. This is a non-blocking function, which returns right away. After the CAN Message is received, the receive callback function is called.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	Pointer to flexcan_edma_handle_t structure.
<i>pFifoXfer</i>	FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_FLEXCAN_Rx-FifoBusy</i>	Previous transfer ongoing.

31.3.5.5 void FLEXCAN_TransferAbortReceiveFifoEDMA (CAN_Type * *base*, flexcan_edma_handle_t * *handle*)

This function aborts the receive Legacy/Enhanced Rx FIFO process which used eDMA.

Parameters

<i>base</i>	FlexCAN peripheral base address.
<i>handle</i>	Pointer to flexcan_edma_handle_t structure.



Chapter 32

FlexIO: FlexIO Driver

32.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FlexIO drivers for the FlexIO module of MCUXpresso SDK devices.

Modules

- [FlexIO Camera Driver](#)
- [FlexIO Driver](#)
- [FlexIO I2C Master Driver](#)
- [FlexIO I2S Driver](#)
- [FlexIO MCU Interface LCD Driver](#)
- [FlexIO SPI Driver](#)
- [FlexIO UART Driver](#)

32.2 FlexIO Driver

32.2.1 Overview

Data Structures

- struct `flexio_config_t`
Define FlexIO user configuration structure. [More...](#)
- struct `flexio_timer_config_t`
Define FlexIO timer configuration structure. [More...](#)
- struct `flexio_shifter_config_t`
Define FlexIO shifter configuration structure. [More...](#)

Macros

- #define `FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x)` `((uint32_t)(x) << 1U)`
Calculate FlexIO timer trigger.

Typedefs

- typedef void(* `flexio_isr_t`)(void *base, void *handle)
typedef for FlexIO simulated driver interrupt handler.

Enumerations

- enum `flexio_timer_trigger_polarity_t` {
 `kFLEXIO_TimerTriggerPolarityActiveHigh` = 0x0U,
 `kFLEXIO_TimerTriggerPolarityActiveLow` = 0x1U }
Define time of timer trigger polarity.
- enum `flexio_timer_trigger_source_t` {
 `kFLEXIO_TimerTriggerSourceExternal` = 0x0U,
 `kFLEXIO_TimerTriggerSourceInternal` = 0x1U }
Define type of timer trigger source.
- enum `flexio_pin_config_t` {
 `kFLEXIO_PinConfigOutputDisabled` = 0x0U,
 `kFLEXIO_PinConfigOpenDrainOrBidirection` = 0x1U,
 `kFLEXIO_PinConfigBidirectionOutputData` = 0x2U,
 `kFLEXIO_PinConfigOutput` = 0x3U }
Define type of timer/shifter pin configuration.
- enum `flexio_pin_polarity_t` {
 `kFLEXIO_PinActiveHigh` = 0x0U,
 `kFLEXIO_PinActiveLow` = 0x1U }
Definition of pin polarity.

- enum flexio_timer_mode_t {
kFLEXIO_TimerModeDisabled = 0x0U,
kFLEXIO_TimerModeDual8BitBaudBit = 0x1U,
kFLEXIO_TimerModeDual8BitPWM = 0x2U,
kFLEXIO_TimerModeSingle16Bit = 0x3U }
Define type of timer work mode.
- enum flexio_timer_output_t {
kFLEXIO_TimerOutputOneNotAffectedByReset = 0x0U,
kFLEXIO_TimerOutputZeroNotAffectedByReset = 0x1U,
kFLEXIO_TimerOutputOneAffectedByReset = 0x2U,
kFLEXIO_TimerOutputZeroAffectedByReset = 0x3U }
Define type of timer initial output or timer reset condition.
- enum flexio_timer_decrement_source_t {
kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput = 0x0U,
kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput = 0x1U,
kFLEXIO_TimerDecSrcOnPinInputShiftPinInput = 0x2U,
kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput = 0x3U }
Define type of timer decrement.
- enum flexio_timer_reset_condition_t {
kFLEXIO_TimerResetNever = 0x0U,
kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput = 0x2U,
kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput = 0x3U,
kFLEXIO_TimerResetOnTimerPinRisingEdge = 0x4U,
kFLEXIO_TimerResetOnTimerTriggerRisingEdge = 0x6U,
kFLEXIO_TimerResetOnTimerTriggerBothEdge = 0x7U }
Define type of timer reset condition.
- enum flexio_timer_disable_condition_t {
kFLEXIO_TimerDisableNever = 0x0U,
kFLEXIO_TimerDisableOnPreTimerDisable = 0x1U,
kFLEXIO_TimerDisableOnTimerCompare = 0x2U,
kFLEXIO_TimerDisableOnTimerCompareTriggerLow = 0x3U,
kFLEXIO_TimerDisableOnPinBothEdge = 0x4U,
kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh = 0x5U,
kFLEXIO_TimerDisableOnTriggerFallingEdge = 0x6U }
Define type of timer disable condition.
- enum flexio_timer_enable_condition_t {
kFLEXIO_TimerEnabledAlways = 0x0U,
kFLEXIO_TimerEnableOnPrevTimerEnable = 0x1U,
kFLEXIO_TimerEnableOnTriggerHigh = 0x2U,
kFLEXIO_TimerEnableOnTriggerHighPinHigh = 0x3U,
kFLEXIO_TimerEnableOnPinRisingEdge = 0x4U,
kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh = 0x5U,
kFLEXIO_TimerEnableOnTriggerRisingEdge = 0x6U,
kFLEXIO_TimerEnableOnTriggerBothEdge = 0x7U }
Define type of timer enable condition.
- enum flexio_timer_stop_bit_condition_t {

```

kFLEXIO_TimerStopBitDisabled = 0x0U,
kFLEXIO_TimerStopBitEnableOnTimerCompare = 0x1U,
kFLEXIO_TimerStopBitEnableOnTimerDisable = 0x2U,
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable = 0x3U }

```

Define type of timer stop bit generate condition.

- enum flexio_timer_start_bit_condition_t {
kFLEXIO_TimerStartBitDisabled = 0x0U,
kFLEXIO_TimerStartBitEnabled = 0x1U }

Define type of timer start bit generate condition.

- enum flexio_shifter_timer_polarity_t {
kFLEXIO_ShifterTimerPolarityOnPositive = 0x0U,
kFLEXIO_ShifterTimerPolarityOnNegative = 0x1U }

Define type of timer polarity for shifter control.

- enum flexio_shifter_mode_t {
kFLEXIO_ShifterDisabled = 0x0U,
kFLEXIO_ShifterModeReceive = 0x1U,
kFLEXIO_ShifterModeTransmit = 0x2U,
kFLEXIO_ShifterModeMatchStore = 0x4U,
kFLEXIO_ShifterModeMatchContinuous = 0x5U,
kFLEXIO_ShifterModeState = 0x6U,
kFLEXIO_ShifterModeLogic = 0x7U }

Define type of shifter working mode.

- enum flexio_shifter_input_source_t {
kFLEXIO_ShifterInputFromPin = 0x0U,
kFLEXIO_ShifterInputFromNextShifterOutput = 0x1U }

Define type of shifter input source.

- enum flexio_shifter_stop_bit_t {
kFLEXIO_ShifterStopBitDisable = 0x0U,
kFLEXIO_ShifterStopBitLow = 0x2U,
kFLEXIO_ShifterStopBitHigh = 0x3U }

Define of STOP bit configuration.

- enum flexio_shifter_start_bit_t {
kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable = 0x0U,
kFLEXIO_ShifterStartBitDisabledLoadDataOnShift = 0x1U,
kFLEXIO_ShifterStartBitLow = 0x2U,
kFLEXIO_ShifterStartBitHigh = 0x3U }

Define type of START bit configuration.

- enum flexio_shifter_buffer_type_t {
kFLEXIO_ShifterBuffer = 0x0U,
kFLEXIO_ShifterBufferBitSwapped = 0x1U,
kFLEXIO_ShifterBufferByteSwapped = 0x2U,
kFLEXIO_ShifterBufferBitByteSwapped = 0x3U,
kFLEXIO_ShifterBufferNibbleByteSwapped = 0x4U,
kFLEXIO_ShifterBufferHalfWordSwapped = 0x5U,
kFLEXIO_ShifterBufferNibbleSwapped = 0x6U }

Define FlexIO shifter buffer type.

Variables

- FLEXIO_Type *const [s_flexioBases](#) []
Pointers to flexio bases for each instance.
- const clock_ip_name_t [s_flexioClocks](#) []
Pointers to flexio clocks for each instance.

Driver version

- #define [FSL_FLEXIO_DRIVER_VERSION](#) (MAKE_VERSION(2, 0, 4))
FlexIO driver version.

FlexIO Initialization and De-initialization

- void [FLEXIO_GetDefaultConfig](#) (flexio_config_t *userConfig)
Gets the default configuration to configure the FlexIO module.
- void [FLEXIO_Init](#) (FLEXIO_Type *base, const flexio_config_t *userConfig)
Configures the FlexIO with a FlexIO configuration.
- void [FLEXIO_Deinit](#) (FLEXIO_Type *base)
Gates the FlexIO clock.
- uint32_t [FLEXIO_GetInstance](#) (FLEXIO_Type *base)
Get instance number for FLEXIO module.

FlexIO Basic Operation

- void [FLEXIO_Reset](#) (FLEXIO_Type *base)
Resets the FlexIO module.
- static void [FLEXIO_Enable](#) (FLEXIO_Type *base, bool enable)
Enables the FlexIO module operation.
- static uint32_t [FLEXIO_ReadPinInput](#) (FLEXIO_Type *base)
Reads the input data on each of the FlexIO pins.
- static uint8_t [FLEXIO_GetShifterState](#) (FLEXIO_Type *base)
Gets the current state pointer for state mode use.
- void [FLEXIO_SetShifterConfig](#) (FLEXIO_Type *base, uint8_t index, const flexio_shifter_config_t *shifterConfig)
Configures the shifter with the shifter configuration.
- void [FLEXIO_SetTimerConfig](#) (FLEXIO_Type *base, uint8_t index, const flexio_timer_config_t *timerConfig)
Configures the timer with the timer configuration.

FlexIO Interrupt Operation

- static void [FLEXIO_EnableShifterStatusInterrupts](#) (FLEXIO_Type *base, uint32_t mask)
Enables the shifter status interrupt.
- static void [FLEXIO_DisableShifterStatusInterrupts](#) (FLEXIO_Type *base, uint32_t mask)

- *Disables the shifter status interrupt.*
- static void [FLEXIO_EnableShifterErrorInterrupts](#) (FLEXIO_Type *base, uint32_t mask)
Enables the shifter error interrupt.
- static void [FLEXIO_DisableShifterErrorInterrupts](#) (FLEXIO_Type *base, uint32_t mask)
Disables the shifter error interrupt.
- static void [FLEXIO_EnableTimerStatusInterrupts](#) (FLEXIO_Type *base, uint32_t mask)
Enables the timer status interrupt.
- static void [FLEXIO_DisableTimerStatusInterrupts](#) (FLEXIO_Type *base, uint32_t mask)
Disables the timer status interrupt.

FlexIO Status Operation

- static uint32_t [FLEXIO_GetShifterStatusFlags](#) (FLEXIO_Type *base)
Gets the shifter status flags.
- static void [FLEXIO_ClearShifterStatusFlags](#) (FLEXIO_Type *base, uint32_t mask)
Clears the shifter status flags.
- static uint32_t [FLEXIO_GetShifterErrorFlags](#) (FLEXIO_Type *base)
Gets the shifter error flags.
- static void [FLEXIO_ClearShifterErrorFlags](#) (FLEXIO_Type *base, uint32_t mask)
Clears the shifter error flags.
- static uint32_t [FLEXIO_GetTimerStatusFlags](#) (FLEXIO_Type *base)
Gets the timer status flags.
- static void [FLEXIO_ClearTimerStatusFlags](#) (FLEXIO_Type *base, uint32_t mask)
Clears the timer status flags.

FlexIO DMA Operation

- static void [FLEXIO_EnableShifterStatusDMA](#) (FLEXIO_Type *base, uint32_t mask, bool enable)
Enables/disables the shifter status DMA.
- uint32_t [FLEXIO_GetShifterBufferAddress](#) (FLEXIO_Type *base, [flexio_shifter_buffer_type_t](#) type, uint8_t index)
Gets the shifter buffer address for the DMA transfer usage.
- [status_t](#) [FLEXIO_RegisterHandleIRQ](#) (void *base, void *handle, [flexio_isr_t](#) isr)
Registers the handle and the interrupt handler for the FlexIO-simulated peripheral.
- [status_t](#) [FLEXIO_UnregisterHandleIRQ](#) (void *base)
Unregisters the handle and the interrupt handler for the FlexIO-simulated peripheral.

32.2.2 Data Structure Documentation

32.2.2.1 struct flexio_config_t

Data Fields

- bool [enableFlexio](#)
Enable/disable FlexIO module.
- bool [enableInDoze](#)

- *Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)
Enable/disable FlexIO operation in debug mode.
- bool [enableFastAccess](#)
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.

Field Documentation

(1) bool flexio_config_t::enableFastAccess

32.2.2.2 struct flexio_timer_config_t

Data Fields

- uint32_t [triggerSelect](#)
The internal trigger selection number using MACROs.
- [flexio_timer_trigger_polarity_t](#) [triggerPolarity](#)
Trigger Polarity.
- [flexio_timer_trigger_source_t](#) [triggerSource](#)
Trigger Source, internal (see 'trgsel') or external.
- [flexio_pin_config_t](#) [pinConfig](#)
Timer Pin Configuration.
- uint32_t [pinSelect](#)
Timer Pin number Select.
- [flexio_pin_polarity_t](#) [pinPolarity](#)
Timer Pin Polarity.
- [flexio_timer_mode_t](#) [timerMode](#)
Timer work Mode.
- [flexio_timer_output_t](#) [timerOutput](#)
Configures the initial state of the Timer Output and whether it is affected by the Timer reset.
- [flexio_timer_decrement_source_t](#) [timerDecrement](#)
Configures the source of the Timer decrement and the source of the Shift clock.
- [flexio_timer_reset_condition_t](#) [timerReset](#)
Configures the condition that causes the timer counter (and optionally the timer output) to be reset.
- [flexio_timer_disable_condition_t](#) [timerDisable](#)
Configures the condition that causes the Timer to be disabled and stop decrementing.
- [flexio_timer_enable_condition_t](#) [timerEnable](#)
Configures the condition that causes the Timer to be enabled and start decrementing.
- [flexio_timer_stop_bit_condition_t](#) [timerStop](#)
Timer STOP Bit generation.
- [flexio_timer_start_bit_condition_t](#) [timerStart](#)
Timer STRAT Bit generation.
- uint32_t [timerCompare](#)
Value for Timer Compare N Register.

Field Documentation

- (1) `uint32_t flexio_timer_config_t::triggerSelect`
- (2) `flexio_timer_trigger_polarity_t flexio_timer_config_t::triggerPolarity`
- (3) `flexio_timer_trigger_source_t flexio_timer_config_t::triggerSource`
- (4) `flexio_pin_config_t flexio_timer_config_t::pinConfig`
- (5) `uint32_t flexio_timer_config_t::pinSelect`
- (6) `flexio_pin_polarity_t flexio_timer_config_t::pinPolarity`
- (7) `flexio_timer_mode_t flexio_timer_config_t::timerMode`
- (8) `flexio_timer_output_t flexio_timer_config_t::timerOutput`
- (9) `flexio_timer_decrement_source_t flexio_timer_config_t::timerDecrement`
- (10) `flexio_timer_reset_condition_t flexio_timer_config_t::timerReset`
- (11) `flexio_timer_disable_condition_t flexio_timer_config_t::timerDisable`
- (12) `flexio_timer_enable_condition_t flexio_timer_config_t::timerEnable`
- (13) `flexio_timer_stop_bit_condition_t flexio_timer_config_t::timerStop`
- (14) `flexio_timer_start_bit_condition_t flexio_timer_config_t::timerStart`
- (15) `uint32_t flexio_timer_config_t::timerCompare`

32.2.2.3 struct flexio_shifter_config_t

Data Fields

- `uint32_t timerSelect`
Selects which Timer is used for controlling the logic/shift register and generating the Shift clock.
- `flexio_shifter_timer_polarity_t timerPolarity`
Timer Polarity.
- `flexio_pin_config_t pinConfig`
Shifter Pin Configuration.
- `uint32_t pinSelect`
Shifter Pin number Select.
- `flexio_pin_polarity_t pinPolarity`
Shifter Pin Polarity.
- `flexio_shifter_mode_t shifterMode`
Configures the mode of the Shifter.
- `uint32_t parallelWidth`
Configures the parallel width when using parallel mode.

- `flexio_shifter_input_source_t` `inputSource`
Selects the input source for the shifter.
- `flexio_shifter_stop_bit_t` `shifterStop`
Shifter STOP bit.
- `flexio_shifter_start_bit_t` `shifterStart`
Shifter START bit.

Field Documentation

- (1) `uint32_t flexio_shifter_config_t::timerSelect`
- (2) `flexio_shifter_timer_polarity_t flexio_shifter_config_t::timerPolarity`
- (3) `flexio_pin_config_t flexio_shifter_config_t::pinConfig`
- (4) `uint32_t flexio_shifter_config_t::pinSelect`
- (5) `flexio_pin_polarity_t flexio_shifter_config_t::pinPolarity`
- (6) `flexio_shifter_mode_t flexio_shifter_config_t::shifterMode`
- (7) `uint32_t flexio_shifter_config_t::parallelWidth`
- (8) `flexio_shifter_input_source_t flexio_shifter_config_t::inputSource`
- (9) `flexio_shifter_stop_bit_t flexio_shifter_config_t::shifterStop`
- (10) `flexio_shifter_start_bit_t flexio_shifter_config_t::shifterStart`

32.2.3 Macro Definition Documentation

32.2.3.1 `#define FSL_FLEXIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

32.2.3.2 `#define FLEXIO_TIMER_TRIGGER_SEL_PININPUT(x) ((uint32_t)(x) << 1U)`

32.2.4 Typedef Documentation

32.2.4.1 `typedef void(* flexio_isr_t)(void *base, void *handle)`

32.2.5 Enumeration Type Documentation

32.2.5.1 `enum flexio_timer_trigger_polarity_t`

Enumerator

`kFLEXIO_TimerTriggerPolarityActiveHigh` Active high.
`kFLEXIO_TimerTriggerPolarityActiveLow` Active low.

32.2.5.2 enum flexio_timer_trigger_source_t

Enumerator

kFLEXIO_TimerTriggerSourceExternal External trigger selected.

kFLEXIO_TimerTriggerSourceInternal Internal trigger selected.

32.2.5.3 enum flexio_pin_config_t

Enumerator

kFLEXIO_PinConfigOutputDisabled Pin output disabled.

kFLEXIO_PinConfigOpenDrainOrBidirection Pin open drain or bidirectional output enable.

kFLEXIO_PinConfigBidirectionOutputData Pin bidirectional output data.

kFLEXIO_PinConfigOutput Pin output.

32.2.5.4 enum flexio_pin_polarity_t

Enumerator

kFLEXIO_PinActiveHigh Active high.

kFLEXIO_PinActiveLow Active low.

32.2.5.5 enum flexio_timer_mode_t

Enumerator

kFLEXIO_TimerModeDisabled Timer Disabled.

kFLEXIO_TimerModeDual8BitBaudBit Dual 8-bit counters baud/bit mode.

kFLEXIO_TimerModeDual8BitPWM Dual 8-bit counters PWM mode.

kFLEXIO_TimerModeSingle16Bit Single 16-bit counter mode.

32.2.5.6 enum flexio_timer_output_t

Enumerator

kFLEXIO_TimerOutputOneNotAffectedByReset Logic one when enabled and is not affected by timer reset.

kFLEXIO_TimerOutputZeroNotAffectedByReset Logic zero when enabled and is not affected by timer reset.

kFLEXIO_TimerOutputOneAffectedByReset Logic one when enabled and on timer reset.

kFLEXIO_TimerOutputZeroAffectedByReset Logic zero when enabled and on timer reset.

32.2.5.7 enum flexio_timer_decrement_source_t

Enumerator

- kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput*** Decrement counter on FlexIO clock, Shift clock equals Timer output.
- kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput*** Decrement counter on Trigger input (both edges), Shift clock equals Timer output.
- kFLEXIO_TimerDecSrcOnPinInputShiftPinInput*** Decrement counter on Pin input (both edges), Shift clock equals Pin input.
- kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput*** Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

32.2.5.8 enum flexio_timer_reset_condition_t

Enumerator

- kFLEXIO_TimerResetNever*** Timer never reset.
- kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput*** Timer reset on Timer Pin equal to Timer Output.
- kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput*** Timer reset on Timer Trigger equal to Timer Output.
- kFLEXIO_TimerResetOnTimerPinRisingEdge*** Timer reset on Timer Pin rising edge.
- kFLEXIO_TimerResetOnTimerTriggerRisingEdge*** Timer reset on Trigger rising edge.
- kFLEXIO_TimerResetOnTimerTriggerBothEdge*** Timer reset on Trigger rising or falling edge.

32.2.5.9 enum flexio_timer_disable_condition_t

Enumerator

- kFLEXIO_TimerDisableNever*** Timer never disabled.
- kFLEXIO_TimerDisableOnPreTimerDisable*** Timer disabled on Timer N-1 disable.
- kFLEXIO_TimerDisableOnTimerCompare*** Timer disabled on Timer compare.
- kFLEXIO_TimerDisableOnTimerCompareTriggerLow*** Timer disabled on Timer compare and Trigger Low.
- kFLEXIO_TimerDisableOnPinBothEdge*** Timer disabled on Pin rising or falling edge.
- kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh*** Timer disabled on Pin rising or falling edge provided Trigger is high.
- kFLEXIO_TimerDisableOnTriggerFallingEdge*** Timer disabled on Trigger falling edge.

32.2.5.10 enum flexio_timer_enable_condition_t

Enumerator

- kFLEXIO_TimerEnabledAlways*** Timer always enabled.

kFLEXIO_TimerEnableOnPrevTimerEnable Timer enabled on Timer N-1 enable.
kFLEXIO_TimerEnableOnTriggerHigh Timer enabled on Trigger high.
kFLEXIO_TimerEnableOnTriggerHighPinHigh Timer enabled on Trigger high and Pin high.
kFLEXIO_TimerEnableOnPinRisingEdge Timer enabled on Pin rising edge.
kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh Timer enabled on Pin rising edge and Trigger high.
kFLEXIO_TimerEnableOnTriggerRisingEdge Timer enabled on Trigger rising edge.
kFLEXIO_TimerEnableOnTriggerBothEdge Timer enabled on Trigger rising or falling edge.

32.2.5.11 enum flexio_timer_stop_bit_condition_t

Enumerator

kFLEXIO_TimerStopBitDisabled Stop bit disabled.
kFLEXIO_TimerStopBitEnableOnTimerCompare Stop bit is enabled on timer compare.
kFLEXIO_TimerStopBitEnableOnTimerDisable Stop bit is enabled on timer disable.
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable Stop bit is enabled on timer compare and timer disable.

32.2.5.12 enum flexio_timer_start_bit_condition_t

Enumerator

kFLEXIO_TimerStartBitDisabled Start bit disabled.
kFLEXIO_TimerStartBitEnabled Start bit enabled.

32.2.5.13 enum flexio_shifter_timer_polarity_t

Enumerator

kFLEXIO_ShifterTimerPolarityOnPositive Shift on positive edge of shift clock.
kFLEXIO_ShifterTimerPolarityOnNegative Shift on negative edge of shift clock.

32.2.5.14 enum flexio_shifter_mode_t

Enumerator

kFLEXIO_ShifterDisabled Shifter is disabled.
kFLEXIO_ShifterModeReceive Receive mode.
kFLEXIO_ShifterModeTransmit Transmit mode.
kFLEXIO_ShifterModeMatchStore Match store mode.
kFLEXIO_ShifterModeMatchContinuous Match continuous mode.

kFLEXIO_ShifterModeState SHIFTBUF contents are used for storing programmable state attributes.

kFLEXIO_ShifterModeLogic SHIFTBUF contents are used for implementing programmable logic look up table.

32.2.5.15 enum flexio_shifter_input_source_t

Enumerator

kFLEXIO_ShifterInputFromPin Shifter input from pin.

kFLEXIO_ShifterInputFromNextShifterOutput Shifter input from Shifter N+1.

32.2.5.16 enum flexio_shifter_stop_bit_t

Enumerator

kFLEXIO_ShifterStopBitDisable Disable shifter stop bit.

kFLEXIO_ShifterStopBitLow Set shifter stop bit to logic low level.

kFLEXIO_ShifterStopBitHigh Set shifter stop bit to logic high level.

32.2.5.17 enum flexio_shifter_start_bit_t

Enumerator

kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable Disable shifter start bit, transmitter loads data on enable.

kFLEXIO_ShifterStartBitDisabledLoadDataOnShift Disable shifter start bit, transmitter loads data on first shift.

kFLEXIO_ShifterStartBitLow Set shifter start bit to logic low level.

kFLEXIO_ShifterStartBitHigh Set shifter start bit to logic high level.

32.2.5.18 enum flexio_shifter_buffer_type_t

Enumerator

kFLEXIO_ShifterBuffer Shifter Buffer N Register.

kFLEXIO_ShifterBufferBitSwapped Shifter Buffer N Bit Byte Swapped Register.

kFLEXIO_ShifterBufferByteSwapped Shifter Buffer N Byte Swapped Register.

kFLEXIO_ShifterBufferBitByteSwapped Shifter Buffer N Bit Swapped Register.

kFLEXIO_ShifterBufferNibbleByteSwapped Shifter Buffer N Nibble Byte Swapped Register.

kFLEXIO_ShifterBufferHalfWordSwapped Shifter Buffer N Half Word Swapped Register.

kFLEXIO_ShifterBufferNibbleSwapped Shifter Buffer N Nibble Swapped Register.

32.2.6 Function Documentation

32.2.6.1 void FLEXIO_GetDefaultConfig (flexio_config_t * *userConfig*)

The configuration can be used directly to call the FLEXIO_Configure().

Example:

```
flexio_config_t config;
FLEXIO_GetDefaultConfig(&config);
```

Parameters

<i>userConfig</i>	pointer to flexio_config_t structure
-------------------	--

32.2.6.2 void FLEXIO_Init (FLEXIO_Type * *base*, const flexio_config_t * *userConfig*)

The configuration structure can be filled by the user or be set with default values by [FLEXIO_GetDefaultConfig\(\)](#).

Example

```
flexio_config_t config = {
    .enableFlexio = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false
};
FLEXIO_Configure(base, &config);
```

Parameters

<i>base</i>	FlexIO peripheral base address
<i>userConfig</i>	pointer to flexio_config_t structure

32.2.6.3 void FLEXIO_Deinit (FLEXIO_Type * *base*)

Call this API to stop the FlexIO clock.

Note

After calling this API, call the FLEXIO_Init to use the FlexIO module.

Parameters

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

32.2.6.4 uint32_t FLEXIO_GetInstance (FLEXIO_Type * *base*)

Parameters

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

32.2.6.5 void FLEXIO_Reset (FLEXIO_Type * *base*)

Parameters

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

32.2.6.6 static void FLEXIO_Enable (FLEXIO_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	FlexIO peripheral base address
<i>enable</i>	true to enable, false to disable.

32.2.6.7 static uint32_t FLEXIO_ReadPinInput (FLEXIO_Type * *base*) [inline], [static]

Parameters

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

Returns

FlexIO pin input data

32.2.6.8 static uint8_t FLEXIO_GetShifterState (FLEXIO_Type * *base*) [inline], [static]

Parameters

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

Returns

current State pointer

32.2.6.9 void FLEXIO_SetShifterConfig (FLEXIO_Type * *base*, uint8_t *index*, const flexio_shifter_config_t * *shifterConfig*)

The configuration structure covers both the SHIFTCTL and SHIFTCFG registers. To configure the shifter to the proper mode, select which timer controls the shifter to shift, whether to generate start bit/stop bit, and the polarity of start bit and stop bit.

Example

```
flexio_shifter_config_t config = {
    .timerSelect = 0,
    .timerPolarity = kFLEXIO_ShifterTimerPolarityOnPositive,
    .pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
    .pinPolarity = kFLEXIO_PinActiveLow,
    .shifterMode = kFLEXIO_ShifterModeTransmit,
    .inputSource = kFLEXIO_ShifterInputFromPin,
    .shifterStop = kFLEXIO_ShifterStopBitHigh,
    .shifterStart = kFLEXIO_ShifterStartBitLow
};
FLEXIO_SetShifterConfig(base, &config);
```

Parameters

<i>base</i>	FlexIO peripheral base address
<i>index</i>	Shifter index
<i>shifterConfig</i>	Pointer to flexio_shifter_config_t structure

32.2.6.10 void FLEXIO_SetTimerConfig (FLEXIO_Type * *base*, uint8_t *index*, const flexio_timer_config_t * *timerConfig*)

The configuration structure covers both the TIMCTL and TIMCFG registers. To configure the timer to the proper mode, select trigger source for timer and the timer pin output and the timing for timer.

Example

```
flexio_timer_config_t config = {
    .triggerSelect = FLEXIO_TIMER_TRIGGER_SEL_SHIFtnSTAT(0),
    .triggerPolarity = kFLEXIO_TimerTriggerPolarityActiveLow,
    .triggerSource = kFLEXIO_TimerTriggerSourceInternal,
```

```

.pinConfig = kFLEXIO_PinConfigOpenDrainOrBidirection,
.pinSelect = 0,
.pinPolarity = kFLEXIO_PinActiveHigh,
.timerMode = kFLEXIO_TimerModeDual8BitBaudBit,
.timerOutput = kFLEXIO_TimerOutputZeroNotAffectedByReset,
.timerDecrement = kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput
,
.timerReset = kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput,
.timerDisable = kFLEXIO_TimerDisableOnTimerCompare,
.timerEnable = kFLEXIO_TimerEnableOnTriggerHigh,
.timerStop = kFLEXIO_TimerStopBitEnableOnTimerDisable,
.timerStart = kFLEXIO_TimerStartBitEnabled
};
FLEXIO_SetTimerConfig(base, &config);

```

Parameters

<i>base</i>	FlexIO peripheral base address
<i>index</i>	Timer index
<i>timerConfig</i>	Pointer to the flexio_timer_config_t structure

32.2.6.11 static void FLEXIO_EnableShifterStatusInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt generates when the corresponding SSF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

32.2.6.12 static void FLEXIO_DisableShifterStatusInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

Note

For multiple shifter status interrupt enable, for example, two shifter status enable, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

32.2.6.13 static void FLEXIO_EnableShifterErrorInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt generates when the corresponding SEF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

32.2.6.14 static void FLEXIO_DisableShifterErrorInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt won't generate when the corresponding SEF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

Note

For multiple shifter error interrupt enable, for example, two shifter error enable, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

32.2.6.15 `static void FLEXIO_EnableTimerStatusInterrupts (FLEXIO_Type * base,
uint32_t mask) [inline], [static]`

The interrupt generates when the corresponding SSF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The timer status mask which can be calculated by $(1 \ll \text{timer index})$

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) \mid (1 \ll \text{timer index1}))$

32.2.6.16 static void FLEXIO_DisableTimerStatusInterrupts (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

The interrupt won't generate when the corresponding SSF is set.

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The timer status mask which can be calculated by $(1 \ll \text{timer index})$

Note

For multiple timer status interrupt enable, for example, two timer status enable, can calculate the mask by using $((1 \ll \text{timer index0}) \mid (1 \ll \text{timer index1}))$

32.2.6.17 static uint32_t FLEXIO_GetShifterStatusFlags (FLEXIO_Type * *base*) [inline], [static]

Parameters

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

Returns

Shifter status flags

32.2.6.18 static void FLEXIO_ClearShifterStatusFlags (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

Note

For clearing multiple shifter status flags, for example, two shifter status flags, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

32.2.6.19 static uint32_t FLEXIO_GetShifterErrorFlags (FLEXIO_Type * *base*) [inline], [static]

Parameters

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

Returns

Shifter error flags

32.2.6.20 static void FLEXIO_ClearShifterErrorFlags (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter error mask which can be calculated by $(1 \ll \text{shifter index})$

Note

For clearing multiple shifter error flags, for example, two shifter error flags, can calculate the mask by using $((1 \ll \text{shifter index0}) \mid (1 \ll \text{shifter index1}))$

32.2.6.21 static uint32_t FLEXIO_GetTimerStatusFlags (FLEXIO_Type * *base*) [inline], [static]

Parameters

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

Returns

Timer status flags

32.2.6.22 static void FLEXIO_ClearTimerStatusFlags (FLEXIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The timer status mask which can be calculated by $(1 \ll \text{timer index})$

Note

For clearing multiple timer status flags, for example, two timer status flags, can calculate the mask by using $((1 \ll \text{timer index0}) | (1 \ll \text{timer index1}))$

32.2.6.23 static void FLEXIO_EnableShifterStatusDMA (FLEXIO_Type * *base*, uint32_t *mask*, bool *enable*) [inline], [static]

The DMA request generates when the corresponding SSF is set.

Note

For multiple shifter status DMA enables, for example, calculate the mask by using $((1 \ll \text{shifter index0}) | (1 \ll \text{shifter index1}))$

Parameters

<i>base</i>	FlexIO peripheral base address
<i>mask</i>	The shifter status mask which can be calculated by $(1 \ll \text{shifter index})$

<i>enable</i>	True to enable, false to disable.
---------------	-----------------------------------

32.2.6.24 uint32_t FLEXIO_GetShifterBufferAddress (FLEXIO_Type * *base*, flexio_shifter_buffer_type_t *type*, uint8_t *index*)

Parameters

<i>base</i>	FlexIO peripheral base address
<i>type</i>	Shifter type of flexio_shifter_buffer_type_t
<i>index</i>	Shifter index

Returns

Corresponding shifter buffer index

32.2.6.25 status_t FLEXIO_RegisterHandleIRQ (void * *base*, void * *handle*, flexio_isr_t *isr*)

Parameters

<i>base</i>	Pointer to the FlexIO simulated peripheral type.
<i>handle</i>	Pointer to the handler for FlexIO simulated peripheral.
<i>isr</i>	FlexIO simulated peripheral interrupt handler.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

32.2.6.26 status_t FLEXIO_UnregisterHandleIRQ (void * *base*)

Parameters

<i>base</i>	Pointer to the FlexIO simulated peripheral type.
-------------	--

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

32.2.7 Variable Documentation

32.2.7.1 FLEXIO_Type* const s_flexioBases[]

32.2.7.2 const clock_ip_name_t s_flexioClocks[]

32.3 FlexIO Camera Driver

32.3.1 Overview

The MCUXpresso SDK provides a driver for the camera function using Flexible I/O.

FlexIO Camera driver includes functional APIs and eDMA transactional APIs. Functional APIs target low level APIs. Users can use functional APIs for FlexIO Camera initialization/configuration/operation purpose. Using the functional API requires knowledge of the FlexIO Camera peripheral and how to organize functional APIs to meet the requirements of the application. All functional API use the [FLEXIO_CAMERA_Type](#) * as the first parameter. FlexIO Camera functional operation groups provide the functional APIs set.

eDMA transactional APIs target high-level APIs. Users can use the transactional API to enable the peripheral quickly and can also use in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_camera_edma_handle_t` as the second parameter. Users need to initialize the handle by calling the [FLEXIO_CAMERA_TransferCreateHandleEDMA\(\)](#) API.

eDMA transactional APIs support asynchronous receive. This means that the functions [FLEXIO_CAMERA_TransferReceiveEDMA\(\)](#) set up an interrupt for data receive. When the receive is complete, the upper layer is notified through a callback function with the status `kStatus_FLEXIO_CAMERA_RxIdle`.

32.3.2 Typical use case

32.3.2.1 FlexIO Camera Receive using eDMA method

```
volatile uint32_t isEDMAGetOnePictureFinish = false;
edma_handle_t g_edmaHandle;
flexio_camera_edma_handle_t g_cameraEdmaHandle;
edma_config_t edmaConfig;
FLEXIO_CAMERA_Type g_FlexioCameraDevice = {.flexioBase = FLEXIO0,
                                             .datPinStartIdx = 24U, /* fxio_pin 24 -31 are used. */
                                             .pclkPinIdx = 1U,      /* fxio_pin 1 is used as pclk pin. */
                                             .hrefPinIdx = 18U,      /* flexio_pin 18 is used as href pin. */
                                             .shifterStartIdx = 0U, /* Shifter 0 = 7 are used. */
                                             .shifterCount = 8U,
                                             .timerIdx = 0U};

flexio_camera_config_t cameraConfig;

/* Configure DMAMUX */
DMAMUX_Init(DMAMUX0);
/* Configure DMA */
EDMA_GetDefaultConfig(&edmaConfig);
EDMA_Init(DMA0, &edmaConfig);

DMAMUX_SetSource(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF, (g_FlexioCameraDevice.
    shifterStartIdx + 1U));
DMAMUX_EnableChannel(DMAMUX0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);
EDMA_CreateHandle(&g_edmaHandle, DMA0, DMA_CHN_FLEXIO_TO_FRAMEBUFF);

FLEXIO_CAMERA_GetDefaultConfig(&cameraConfig);
FLEXIO_CAMERA_Init(&g_FlexioCameraDevice, &cameraConfig);
/* Clear all the flag. */
```

```

FLEXIO_CAMERA_ClearStatusFlags(&g_FlexioCameraDevice,
                                kFLEXIO_CAMERA_RxDataRegFullFlag |
                                kFLEXIO_CAMERA_RxErrorFlag);
FLEXIO_ClearTimerStatusFlags(FLEXIO0, 0xFF);
FLEXIO_CAMERA_TransferCreateHandleEDMA(&g_FlexioCameraDevice, &
                                         g_cameraEdmaHandle, FLEXIO_CAMERA_UserCallback, NULL,
                                         &g_edmaHandle);
cameraTransfer.dataAddress = (uint32_t)u16CameraFrameBuffer;
cameraTransfer.dataNum = sizeof(u16CameraFrameBuffer);
FLEXIO_CAMERA_TransferReceiveEDMA(&g_FlexioCameraDevice, &
                                   g_cameraEdmaHandle, &cameraTransfer);
while (!(isEDMAGetOnePictureFinish))
{
    ;
}

/* A callback function is also needed */
void FLEXIO_CAMERA_UserCallback(FLEXIO_CAMERA_Type *base,
                                flexio_camera_edma_handle_t *handle,
                                status_t status,
                                void *userData)
{
    userData = userData;
    /* eDMA Transfer finished */
    if (kStatus_FLEXIO_CAMERA_RxIdle == status)
    {
        isEDMAGetOnePictureFinish = true;
    }
}

```

Modules

- [FlexIO eDMA Camera Driver](#)

Data Structures

- struct [FLEXIO_CAMERA_Type](#)
Define structure of configuring the FlexIO Camera device. [More...](#)
- struct [flexio_camera_config_t](#)
Define FlexIO Camera user configuration structure. [More...](#)
- struct [flexio_camera_transfer_t](#)
Define FlexIO Camera transfer structure. [More...](#)

Macros

- #define [FLEXIO_CAMERA_PARALLEL_DATA_WIDTH](#) (8U)
Define the Camera CPI interface is constantly 8-bit width.

Enumerations

- enum {
 [kStatus_FLEXIO_CAMERA_RxBusy](#) = MAKE_STATUS(kStatusGroup_FLEXIO_CAMERA,

```

0),
kStatus_FLEXIO_CAMERA_RxIdle = MAKE_STATUS(kStatusGroup_FLEXIO_CAMERA, 1)
}
    Error codes for the Camera driver.
• enum _flexio_camera_status_flags {
    kFLEXIO_CAMERA_RxDataRegFullFlag = 0x1U,
    kFLEXIO_CAMERA_RxErrorFlag = 0x2U }
    Define FlexIO Camera status mask.

```

Driver version

- #define FSL_FLEXIO_CAMERA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))
FlexIO Camera driver version 2.1.3.

Initialization and configuration

- void FLEXIO_CAMERA_Init (FLEXIO_CAMERA_Type *base, const flexio_camera_config_t *config)
Ungates the FlexIO clock, resets the FlexIO module, and configures the FlexIO Camera.
- void FLEXIO_CAMERA_Deinit (FLEXIO_CAMERA_Type *base)
Resets the FLEXIO_CAMERA shifer and timer config.
- void FLEXIO_CAMERA_GetDefaultConfig (flexio_camera_config_t *config)
Gets the default configuration to configure the FlexIO Camera.
- static void FLEXIO_CAMERA_Enable (FLEXIO_CAMERA_Type *base, bool enable)
Enables/disables the FlexIO Camera module operation.

Status

- uint32_t FLEXIO_CAMERA_GetStatusFlags (FLEXIO_CAMERA_Type *base)
Gets the FlexIO Camera status flags.
- void FLEXIO_CAMERA_ClearStatusFlags (FLEXIO_CAMERA_Type *base, uint32_t mask)
Clears the receive buffer full flag manually.

Interrupts

- void FLEXIO_CAMERA_EnableInterrupt (FLEXIO_CAMERA_Type *base)
Switches on the interrupt for receive buffer full event.
- void FLEXIO_CAMERA_DisableInterrupt (FLEXIO_CAMERA_Type *base)
Switches off the interrupt for receive buffer full event.

DMA support

- static void FLEXIO_CAMERA_EnableRxDMA (FLEXIO_CAMERA_Type *base, bool enable)

Enables/disables the FlexIO Camera receive DMA.

- static uint32_t [FLEXIO_CAMERA_GetRxBufferAddress](#) ([FLEXIO_CAMERA_Type](#) *base)
Gets the data from the receive buffer.

32.3.3 Data Structure Documentation

32.3.3.1 struct FLEXIO_CAMERA_Type

Data Fields

- [FLEXIO_Type](#) * [flexioBase](#)
FlexIO module base address.
- uint32_t [datPinStartIdx](#)
First data pin (D0) index for flexio_camera.
- uint32_t [pclkPinIdx](#)
Pixel clock pin (PCLK) index for flexio_camera.
- uint32_t [hrefPinIdx](#)
Horizontal sync pin (HREF) index for flexio_camera.
- uint32_t [shifterStartIdx](#)
First shifter index used for flexio_camera data FIFO.
- uint32_t [shifterCount](#)
The count of shifters that are used as flexio_camera data FIFO.
- uint32_t [timerIdx](#)
Timer index used for flexio_camera in FlexIO.

Field Documentation

(1) [FLEXIO_Type](#)* [FLEXIO_CAMERA_Type::flexioBase](#)

(2) [uint32_t](#) [FLEXIO_CAMERA_Type::datPinStartIdx](#)

Then the successive following [FLEXIO_CAMERA_DATA_WIDTH](#)-1 pins are used as D1-D7.

(3) [uint32_t](#) [FLEXIO_CAMERA_Type::pclkPinIdx](#)

(4) [uint32_t](#) [FLEXIO_CAMERA_Type::hrefPinIdx](#)

(5) [uint32_t](#) [FLEXIO_CAMERA_Type::shifterStartIdx](#)

(6) [uint32_t](#) [FLEXIO_CAMERA_Type::shifterCount](#)

(7) [uint32_t](#) [FLEXIO_CAMERA_Type::timerIdx](#)

32.3.3.2 struct flexio_camera_config_t

Data Fields

- bool [enablecamera](#)
Enable/disable FlexIO Camera TX & RX.

- bool [enableInDoze](#)
Enable/disable FlexIO operation in doze mode.
- bool [enableInDebug](#)
Enable/disable FlexIO operation in debug mode.
- bool [enableFastAccess](#)
*Enable/disable fast access to FlexIO registers,
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*

Field Documentation

(1) bool flexio_camera_config_t::enablecamera

(2) bool flexio_camera_config_t::enableFastAccess

32.3.3.3 struct flexio_camera_transfer_t

Data Fields

- uint32_t [dataAddress](#)
Transfer buffer.
- uint32_t [dataNum](#)
Transfer num.

32.3.4 Macro Definition Documentation

32.3.4.1 #define FSL_FLEXIO_CAMERA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

32.3.4.2 #define FLEXIO_CAMERA_PARALLEL_DATA_WIDTH (8U)

32.3.5 Enumeration Type Documentation

32.3.5.1 anonymous enum

Enumerator

kStatus_FLEXIO_CAMERA_RxBusy Receiver is busy.
kStatus_FLEXIO_CAMERA_RxIdle Camera receiver is idle.

32.3.5.2 enum _flexio_camera_status_flags

Enumerator

kFLEXIO_CAMERA_RxDataRegFullFlag Receive buffer full flag.
kFLEXIO_CAMERA_RxErrorFlag Receive buffer error flag.

32.3.6 Function Documentation

32.3.6.1 void FLEXIO_CAMERA_Init (FLEXIO_CAMERA_Type * *base*, const flexio_camera_config_t * *config*)

Parameters

<i>base</i>	Pointer to FLEXIO_CAMERA_Type structure
<i>config</i>	Pointer to flexio_camera_config_t structure

32.3.6.2 void FLEXIO_CAMERA_Deinit (FLEXIO_CAMERA_Type * *base*)

Note

After calling this API, call FLEXIO_CAMERA_Init to use the FlexIO Camera module.

Parameters

<i>base</i>	Pointer to FLEXIO_CAMERA_Type structure
-------------	---

32.3.6.3 void FLEXIO_CAMERA_GetDefaultConfig (flexio_camera_config_t * *config*)

The configuration can be used directly for calling the [FLEXIO_CAMERA_Init\(\)](#). Example:

```
flexio_camera_config_t config;
FLEXIO_CAMERA_GetDefaultConfig(&userConfig);
```

Parameters

<i>config</i>	Pointer to the flexio_camera_config_t structure
---------------	---

32.3.6.4 static void FLEXIO_CAMERA_Enable (FLEXIO_CAMERA_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_CAMERA_Type
<i>enable</i>	True to enable, false does not have any effect.

32.3.6.5 uint32_t FLEXIO_CAMERA_GetStatusFlags (FLEXIO_CAMERA_Type * *base*)

Parameters

<i>base</i>	Pointer to FLEXIO_CAMERA_Type structure
-------------	---

Returns

FlexIO shifter status flags

- FLEXIO_SHIFTSTAT_SSF_MASK
- 0

32.3.6.6 void FLEXIO_CAMERA_ClearStatusFlags (FLEXIO_CAMERA_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to the device.
<i>mask</i>	status flag The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_CAMERA_RxDataRegFullFlag • kFLEXIO_CAMERA_RxErrorFlag

32.3.6.7 void FLEXIO_CAMERA_EnableInterrupt (FLEXIO_CAMERA_Type * *base*)

Parameters

<i>base</i>	Pointer to the device.
-------------	------------------------

32.3.6.8 void FLEXIO_CAMERA_DisableInterrupt (FLEXIO_CAMERA_Type * *base*)

Parameters

<i>base</i>	Pointer to the device.
-------------	------------------------

32.3.6.9 static void FLEXIO_CAMERA_EnableRxDMA (FLEXIO_CAMERA_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to FLEXIO_CAMERA_Type structure
<i>enable</i>	True to enable, false to disable.

The FlexIO Camera mode can't work without the DMA or eDMA support, Usually, it needs at least two DMA or eDMA channels, one for transferring data from Camera, such as 0V7670 to FlexIO buffer, another is for transferring data from FlexIO buffer to LCD.

32.3.6.10 static uint32_t FLEXIO_CAMERA_GetRxBufferAddress (FLEXIO_CAMERA_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Pointer to the device.
-------------	------------------------

Returns

data Pointer to the buffer that keeps the data with count of base->shifterCount .

32.3.7 FlexIO eDMA Camera Driver

32.3.7.1 Overview

Data Structures

- struct `flexio_camera_edma_handle_t`
Camera eDMA handle. [More...](#)

Typedefs

- typedef void(* `flexio_camera_edma_transfer_callback_t`)(`FLEXIO_CAMERA_Type` *base, `flexio_camera_edma_handle_t` *handle, `status_t` status, void *userData)
Camera transfer callback function.

Driver version

- #define `FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 3))
FlexIO Camera EDMA driver version 2.1.3.

eDMA transactional

- `status_t FLEXIO_CAMERA_TransferCreateHandleEDMA` (`FLEXIO_CAMERA_Type` *base, `flexio_camera_edma_handle_t` *handle, `flexio_camera_edma_transfer_callback_t` callback, void *userData, `edma_handle_t` *rxEdmaHandle)
Initializes the Camera handle, which is used in transactional functions.
- `status_t FLEXIO_CAMERA_TransferReceiveEDMA` (`FLEXIO_CAMERA_Type` *base, `flexio_camera_edma_handle_t` *handle, `flexio_camera_transfer_t` *xfer)
Receives data using eDMA.
- void `FLEXIO_CAMERA_TransferAbortReceiveEDMA` (`FLEXIO_CAMERA_Type` *base, `flexio_camera_edma_handle_t` *handle)
Aborts the receive data which used the eDMA.
- `status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA` (`FLEXIO_CAMERA_Type` *base, `flexio_camera_edma_handle_t` *handle, `size_t` *count)
Gets the remaining bytes to be received.

32.3.7.2 Data Structure Documentation

32.3.7.2.1 struct `flexio_camera_edma_handle`

Forward declaration of the handle typedef.

Data Fields

- `flexio_camera_edma_transfer_callback_t` callback

- *Callback function.*
void * [userData](#)
- *Camera callback function parameter.*
size_t [rxSize](#)
- *Total bytes to be received.*
[edma_handle_t](#) * [rxEdmaHandle](#)
- *The eDMA RX channel used.*
uint8_t [nbytes](#)
- *eDMA minor byte transfer count initially configured.*
volatile uint8_t [rxState](#)
- *RX transfer state.*

Field Documentation

- (1) `flexio_camera_edma_transfer_callback_t flexio_camera_edma_handle_t::callback`
- (2) `void* flexio_camera_edma_handle_t::userData`
- (3) `size_t flexio_camera_edma_handle_t::rxSize`
- (4) `edma_handle_t* flexio_camera_edma_handle_t::rxEdmaHandle`
- (5) `uint8_t flexio_camera_edma_handle_t::nbytes`

32.3.7.3 Macro Definition Documentation

32.3.7.3.1 `#define FSL_FLEXIO_CAMERA_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))`

32.3.7.4 Typedef Documentation

32.3.7.4.1 `typedef void(* flexio_camera_edma_transfer_callback_t)(FLEXIO_CAMERA_Type *base, flexio_camera_edma_handle_t *handle, status_t status, void *userData)`

32.3.7.5 Function Documentation

32.3.7.5.1 `status_t FLEXIO_CAMERA_TransferCreateHandleEDMA (FLEXIO_CAMERA_Type * base, flexio_camera_edma_handle_t * handle, flexio_camera_edma_transfer_callback_t callback, void * userData, edma_handle_t * rxEdmaHandle)`

Parameters

<i>base</i>	Pointer to the FLEXIO_CAMERA_Type .
-------------	---

<i>handle</i>	Pointer to flexio_camera_edma_handle_t structure.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.
<i>rxEdmaHandle</i>	User requested DMA handle for RX DMA transfer.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO Camera eDMA type/handle table out of range.

32.3.7.5.2 status_t FLEXIO_CAMERA_TransferReceiveEDMA (FLEXIO_CAMERA_Type * *base*, flexio_camera_edma_handle_t * *handle*, flexio_camera_transfer_t * *xfer*)

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

<i>base</i>	Pointer to the FLEXIO_CAMERA_Type .
<i>handle</i>	Pointer to the flexio_camera_edma_handle_t structure.
<i>xfer</i>	Camera eDMA transfer structure, see flexio_camera_transfer_t .

Return values

<i>kStatus_Success</i>	if succeeded, others failed.
<i>kStatus_CAMERA_Rx-Busy</i>	Previous transfer on going.

32.3.7.5.3 void FLEXIO_CAMERA_TransferAbortReceiveEDMA (FLEXIO_CAMERA_Type * *base*, flexio_camera_edma_handle_t * *handle*)

This function aborts the receive data which used the eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_CAMERA_Type .
-------------	---

<i>handle</i>	Pointer to the flexio_camera_edma_handle_t structure.
---------------	---

32.3.7.5.4 `status_t FLEXIO_CAMERA_TransferGetReceiveCountEDMA (FLEXIO_CAMERA_Type * base, flexio_camera_edma_handle_t * handle, size_t * count)`

This function gets the number of bytes still not received.

Parameters

<i>base</i>	Pointer to the FLEXIO_CAMERA_Type .
<i>handle</i>	Pointer to the flexio_camera_edma_handle_t structure.
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_InvalidArgument</i>	The count parameter is invalid.

32.4 FlexIO I2C Master Driver

32.4.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2C master function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO I2C master driver includes functional APIs and transactional APIs.

Functional APIs target low level APIs. Functional APIs can be used for the FlexIO I2C master initialization/configuration/operation for the optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO I2C master peripheral and how to organize functional APIs to meet the application requirements. The FlexIO I2C master functional operation groups provide the functional APIs set.

Transactional APIs 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 using the functional APIs or accessing the hardware registers.

Transactional APIs support an asynchronous transfer. This means that the functions [FLEXIO_I2C_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_Success` status.

32.4.2 Typical use case

32.4.2.1 FlexIO I2C master transfer using an interrupt method

```
flexio_i2c_master_handle_t g_m_handle;
flexio_i2c_master_config_t masterConfig;
flexio_i2c_master_transfer_t masterXfer;
volatile bool completionFlag = false;
const uint8_t sendData[] = {.....};
FLEXIO_I2C_Type i2cDev;

void FLEXIO_I2C_MasterCallback(FLEXIO_I2C_Type *base, status_t status, void *
    userData)
{
    userData = userData;

    if (kStatus_Success == status)
    {
        completionFlag = true;
    }
}

void main(void)
{
    //...

    FLEXIO_I2C_MasterGetDefaultConfig(&masterConfig);

    FLEXIO_I2C_MasterInit(&i2cDev, &user_config);
    FLEXIO_I2C_MasterTransferCreateHandle(&i2cDev, &g_m_handle,
        FLEXIO_I2C_MasterCallback, NULL);
```

```

// Prepares to send.
masterXfer.slaveAddress = g_accel_address[0];
masterXfer.direction = kI2C_Read;
masterXfer.subaddress = &who_am_i_reg;
masterXfer.subaddressSize = 1;
masterXfer.data = &who_am_i_value;
masterXfer.dataSize = 1;
masterXfer.flags = kI2C_TransferDefaultFlag;

// Sends out.
FLEXIO_I2C_MasterTransferNonBlocking(&i2cDev, &g_m_handle, &
    masterXfer);

// Wait for sending is complete.
while (!completionFlag)
{
}

// ...
}

```

Data Structures

- struct [FLEXIO_I2C_Type](#)
Define FlexIO I2C master access structure typedef. [More...](#)
- struct [flexio_i2c_master_config_t](#)
Define FlexIO I2C master user configuration structure. [More...](#)
- struct [flexio_i2c_master_transfer_t](#)
Define FlexIO I2C master transfer structure. [More...](#)
- struct [flexio_i2c_master_handle_t](#)
Define FlexIO I2C master handle structure. [More...](#)

Macros

- `#define I2C_RETRY_TIMES 0U` /* Define to zero means keep waiting until the flag is assert/deassert. */
Retry times for waiting flag.

Typedefs

- typedef void(* [flexio_i2c_master_transfer_callback_t](#))(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, [status_t](#) status, void *userData)
FlexIO I2C master transfer callback typedef.

Enumerations

- enum {
[kStatus_FLEXIO_I2C_Busy](#) = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 0),
[kStatus_FLEXIO_I2C_Idle](#) = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 1),
[kStatus_FLEXIO_I2C_Nak](#) = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 2),

- ```
kStatus_FLEXIO_I2C_Timeout = MAKE_STATUS(kStatusGroup_FLEXIO_I2C, 3) }
```
- FlexIO I2C transfer status.*
- enum `_flexio_i2c_master_interrupt` {  
`kFLEXIO_I2C_TxEmptyInterruptEnable` = 0x1U,  
`kFLEXIO_I2C_RxFullInterruptEnable` = 0x2U }
- Define FlexIO I2C master interrupt mask.*
- enum `_flexio_i2c_master_status_flags` {  
`kFLEXIO_I2C_TxEmptyFlag` = 0x1U,  
`kFLEXIO_I2C_RxFullFlag` = 0x2U,  
`kFLEXIO_I2C_ReceiveNakFlag` = 0x4U }
- Define FlexIO I2C master status mask.*
- enum `flexio_i2c_direction_t` {  
`kFLEXIO_I2C_Write` = 0x0U,  
`kFLEXIO_I2C_Read` = 0x1U }
- Direction of master transfer.*

## Driver version

- #define `FSL_FLEXIO_I2C_MASTER_DRIVER_VERSION` (`MAKE_VERSION(2, 4, 0)`)

## Initialization and deinitialization

- status\_t `FLEXIO_I2C_CheckForBusyBus` (`FLEXIO_I2C_Type` \*base)  
*Make sure the bus isn't already pulled down.*
- status\_t `FLEXIO_I2C_MasterInit` (`FLEXIO_I2C_Type` \*base, `flexio_i2c_master_config_t` \*masterConfig, uint32\_t srcClock\_Hz)  
*Un gates the FlexIO clock, resets the FlexIO module, and configures the FlexIO I2C hardware configuration.*
- void `FLEXIO_I2C_MasterDeinit` (`FLEXIO_I2C_Type` \*base)  
*De-initializes the FlexIO I2C master peripheral.*
- void `FLEXIO_I2C_MasterGetDefaultConfig` (`flexio_i2c_master_config_t` \*masterConfig)  
*Gets the default configuration to configure the FlexIO module.*
- static void `FLEXIO_I2C_MasterEnable` (`FLEXIO_I2C_Type` \*base, bool enable)  
*Enables/disables the FlexIO module operation.*

## Status

- uint32\_t `FLEXIO_I2C_MasterGetStatusFlags` (`FLEXIO_I2C_Type` \*base)  
*Gets the FlexIO I2C master status flags.*
- void `FLEXIO_I2C_MasterClearStatusFlags` (`FLEXIO_I2C_Type` \*base, uint32\_t mask)  
*Clears the FlexIO I2C master status flags.*

## Interrupts

- void `FLEXIO_I2C_MasterEnableInterrupts` (`FLEXIO_I2C_Type` \*base, uint32\_t mask)

*Enables the FlexIO i2c master interrupt requests.*

- void [FLEXIO\\_I2C\\_MasterDisableInterrupts](#) ([FLEXIO\\_I2C\\_Type](#) \*base, uint32\_t mask)

*Disables the FlexIO I2C master interrupt requests.*

## Bus Operations

- void [FLEXIO\\_I2C\\_MasterSetBaudRate](#) ([FLEXIO\\_I2C\\_Type](#) \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the FlexIO I2C master transfer baudrate.*
- void [FLEXIO\\_I2C\\_MasterStart](#) ([FLEXIO\\_I2C\\_Type](#) \*base, uint8\_t address, [flexio\\_i2c\\_direction\\_t](#) direction)  
*Sends START + 7-bit address to the bus.*
- void [FLEXIO\\_I2C\\_MasterStop](#) ([FLEXIO\\_I2C\\_Type](#) \*base)  
*Sends the stop signal on the bus.*
- void [FLEXIO\\_I2C\\_MasterRepeatedStart](#) ([FLEXIO\\_I2C\\_Type](#) \*base)  
*Sends the repeated start signal on the bus.*
- void [FLEXIO\\_I2C\\_MasterAbortStop](#) ([FLEXIO\\_I2C\\_Type](#) \*base)  
*Sends the stop signal when transfer is still on-going.*
- void [FLEXIO\\_I2C\\_MasterEnableAck](#) ([FLEXIO\\_I2C\\_Type](#) \*base, bool enable)  
*Configures the sent ACK/NAK for the following byte.*
- [status\\_t](#) [FLEXIO\\_I2C\\_MasterSetTransferCount](#) ([FLEXIO\\_I2C\\_Type](#) \*base, uint16\_t count)  
*Sets the number of bytes to be transferred from a start signal to a stop signal.*
- static void [FLEXIO\\_I2C\\_MasterWriteByte](#) ([FLEXIO\\_I2C\\_Type](#) \*base, uint32\_t data)  
*Writes one byte of data to the I2C bus.*
- static uint8\_t [FLEXIO\\_I2C\\_MasterReadByte](#) ([FLEXIO\\_I2C\\_Type](#) \*base)  
*Reads one byte of data from the I2C bus.*
- [status\\_t](#) [FLEXIO\\_I2C\\_MasterWriteBlocking](#) ([FLEXIO\\_I2C\\_Type](#) \*base, const uint8\_t \*txBuff, uint8\_t txSize)  
*Sends a buffer of data in bytes.*
- [status\\_t](#) [FLEXIO\\_I2C\\_MasterReadBlocking](#) ([FLEXIO\\_I2C\\_Type](#) \*base, uint8\_t \*rxBuff, uint8\_t rxSize)  
*Receives a buffer of bytes.*
- [status\\_t](#) [FLEXIO\\_I2C\\_MasterTransferBlocking](#) ([FLEXIO\\_I2C\\_Type](#) \*base, [flexio\\_i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master polling transfer on the I2C bus.*

## Transactional

- [status\\_t](#) [FLEXIO\\_I2C\\_MasterTransferCreateHandle](#) ([FLEXIO\\_I2C\\_Type](#) \*base, [flexio\\_i2c\\_master\\_handle\\_t](#) \*handle, [flexio\\_i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the I2C handle which is used in transactional functions.*
- [status\\_t](#) [FLEXIO\\_I2C\\_MasterTransferNonBlocking](#) ([FLEXIO\\_I2C\\_Type](#) \*base, [flexio\\_i2c\\_master\\_handle\\_t](#) \*handle, [flexio\\_i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a master interrupt non-blocking transfer on the I2C bus.*
- [status\\_t](#) [FLEXIO\\_I2C\\_MasterTransferGetCount](#) ([FLEXIO\\_I2C\\_Type](#) \*base, [flexio\\_i2c\\_master\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the master transfer status during a interrupt non-blocking transfer.*

- void [FLEXIO\\_I2C\\_MasterTransferAbort](#) ([FLEXIO\\_I2C\\_Type](#) \*base, flexio\_i2c\_master\_handle\_t \*handle)  
*Aborts an interrupt non-blocking transfer early.*
- void [FLEXIO\\_I2C\\_MasterTransferHandleIRQ](#) (void \*i2cType, void \*i2cHandle)  
*Master interrupt handler.*

### 32.4.3 Data Structure Documentation

#### 32.4.3.1 struct FLEXIO\_I2C\_Type

##### Data Fields

- [FLEXIO\\_Type](#) \* [flexioBase](#)  
*FlexIO base pointer.*
- uint8\_t [SDAPinIndex](#)  
*Pin select for I2C SDA.*
- uint8\_t [SCLPinIndex](#)  
*Pin select for I2C SCL.*
- uint8\_t [shifterIndex](#) [2]  
*Shifter index used in FlexIO I2C.*
- uint8\_t [timerIndex](#) [3]  
*Timer index used in FlexIO I2C.*
- uint32\_t [baudrate](#)  
*Master transfer baudrate, used to calculate delay time.*

##### Field Documentation

- (1) [FLEXIO\\_Type](#)\* [FLEXIO\\_I2C\\_Type::flexioBase](#)
- (2) uint8\_t [FLEXIO\\_I2C\\_Type::SDAPinIndex](#)
- (3) uint8\_t [FLEXIO\\_I2C\\_Type::SCLPinIndex](#)
- (4) uint8\_t [FLEXIO\\_I2C\\_Type::shifterIndex](#)[2]
- (5) uint8\_t [FLEXIO\\_I2C\\_Type::timerIndex](#)[3]
- (6) uint32\_t [FLEXIO\\_I2C\\_Type::baudrate](#)

#### 32.4.3.2 struct flexio\_i2c\_master\_config\_t

##### Data Fields

- bool [enableMaster](#)  
*Enables the FlexIO I2C peripheral at initialization time.*
- bool [enableInDoze](#)  
*Enable/disable FlexIO operation in doze mode.*
- bool [enableInDebug](#)  
*Enable/disable FlexIO operation in debug mode.*

- bool [enableFastAccess](#)  
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- uint32\_t [baudRate\\_Bps](#)  
Baud rate in Bps.

#### Field Documentation

- (1) bool flexio\_i2c\_master\_config\_t::enableMaster
- (2) bool flexio\_i2c\_master\_config\_t::enableInDoze
- (3) bool flexio\_i2c\_master\_config\_t::enableInDebug
- (4) bool flexio\_i2c\_master\_config\_t::enableFastAccess
- (5) uint32\_t flexio\_i2c\_master\_config\_t::baudRate\_Bps

#### 32.4.3.3 struct flexio\_i2c\_master\_transfer\_t

#### Data Fields

- uint32\_t [flags](#)  
Transfer flag which controls the transfer, reserved for FlexIO I2C.
- uint8\_t [slaveAddress](#)  
7-bit slave address.
- [flexio\\_i2c\\_direction\\_t](#) direction  
Transfer direction, read or write.
- uint32\_t [subaddress](#)  
Sub address.
- uint8\_t [subaddressSize](#)  
Size of command buffer.
- uint8\_t volatile \* [data](#)  
Transfer buffer.
- volatile size\_t [dataSize](#)  
Transfer size.

#### Field Documentation

- (1) uint32\_t flexio\_i2c\_master\_transfer\_t::flags
- (2) uint8\_t flexio\_i2c\_master\_transfer\_t::slaveAddress
- (3) flexio\_i2c\_direction\_t flexio\_i2c\_master\_transfer\_t::direction
- (4) uint32\_t flexio\_i2c\_master\_transfer\_t::subaddress

Transferred MSB first.

- (5) uint8\_t flexio\_i2c\_master\_transfer\_t::subaddressSize

(6) `uint8_t volatile* flexio_i2c_master_transfer_t::data`

(7) `volatile size_t flexio_i2c_master_transfer_t::dataSize`

#### 32.4.3.4 struct flexio\_i2c\_master\_handle

FlexIO I2C master handle typedef.

#### Data Fields

- `flexio_i2c_master_transfer_t transfer`  
*FlexIO I2C master transfer copy.*
- `size_t transferSize`  
*Total bytes to be transferred.*
- `uint8_t state`  
*Transfer state maintained during transfer.*
- `flexio_i2c_master_transfer_callback_t completionCallback`  
*Callback function called at transfer event.*
- `void * userData`  
*Callback parameter passed to callback function.*
- `bool needRestart`  
*Whether master needs to send re-start signal.*

#### Field Documentation

(1) `flexio_i2c_master_transfer_t flexio_i2c_master_handle_t::transfer`

(2) `size_t flexio_i2c_master_handle_t::transferSize`

(3) `uint8_t flexio_i2c_master_handle_t::state`

(4) `flexio_i2c_master_transfer_callback_t flexio_i2c_master_handle_t::completionCallback`

Callback function called at transfer event.

(5) `void* flexio_i2c_master_handle_t::userData`

(6) `bool flexio_i2c_master_handle_t::needRestart`

#### 32.4.4 Macro Definition Documentation

32.4.4.1 `#define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */`

#### 32.4.5 Typedef Documentation

**32.4.5.1** `typedef void(* flexio_i2c_master_transfer_callback_t)(FLEXIO_I2C_Type *base, flexio_i2c_master_handle_t *handle, status_t status, void *userData)`

## 32.4.6 Enumeration Type Documentation

### 32.4.6.1 anonymous enum

Enumerator

*kStatus\_FLEXIO\_I2C\_Busy* I2C is busy doing transfer.  
*kStatus\_FLEXIO\_I2C\_Idle* I2C is busy doing transfer.  
*kStatus\_FLEXIO\_I2C\_Nak* NAK received during transfer.  
*kStatus\_FLEXIO\_I2C\_Timeout* Timeout polling status flags.

### 32.4.6.2 enum flexio\_i2c\_master\_interrupt

Enumerator

*kFLEXIO\_I2C\_TxEmptyInterruptEnable* Tx buffer empty interrupt enable.  
*kFLEXIO\_I2C\_RxFullInterruptEnable* Rx buffer full interrupt enable.

### 32.4.6.3 enum flexio\_i2c\_master\_status\_flags

Enumerator

*kFLEXIO\_I2C\_TxEmptyFlag* Tx shifter empty flag.  
*kFLEXIO\_I2C\_RxFullFlag* Rx shifter full/Transfer complete flag.  
*kFLEXIO\_I2C\_ReceiveNakFlag* Receive NAK flag.

### 32.4.6.4 enum flexio\_i2c\_direction\_t

Enumerator

*kFLEXIO\_I2C\_Write* Master send to slave.  
*kFLEXIO\_I2C\_Read* Master receive from slave.

## 32.4.7 Function Documentation

### 32.4.7.1 status\_t FLEXIO\_I2C\_CheckForBusyBus ( FLEXIO\_I2C\_Type \* base )

Check the FLEXIO pin status to see whether either of SDA and SCL pin is pulled down.

## Parameters

|             |                                                        |
|-------------|--------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.. |
|-------------|--------------------------------------------------------|

## Return values

|                                |  |
|--------------------------------|--|
| <i>kStatus_Success</i>         |  |
| <i>kStatus_FLEXIO_I2C_Busy</i> |  |

**32.4.7.2** `status_t FLEXIO_I2C_MasterInit ( FLEXIO_I2C_Type * base,  
flexio_i2c_master_config_t * masterConfig, uint32_t srcClock_Hz )`

## Example

```
FLEXIO_I2C_Type base = {
 .flexioBase = FLEXIO,
 .SDAPinIndex = 0,
 .SCLPinIndex = 1,
 .shifterIndex = {0,1},
 .timerIndex = {0,1}
};
flexio_i2c_master_config_t config = {
 .enableInDoze = false,
 .enableInDebug = true,
 .enableFastAccess = false,
 .baudRate_Bps = 100000
};
FLEXIO_I2C_MasterInit(base, &config, srcClock_Hz);
```

## Parameters

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <i>base</i>         | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.            |
| <i>masterConfig</i> | Pointer to <a href="#">flexio_i2c_master_config_t</a> structure. |
| <i>srcClock_Hz</i>  | FlexIO source clock in Hz.                                       |

## Return values

|                                |                                                |
|--------------------------------|------------------------------------------------|
| <i>kStatus_Success</i>         | Initialization successful                      |
| <i>kStatus_InvalidArgument</i> | The source clock exceed upper range limitation |

**32.4.7.3** `void FLEXIO_I2C_MasterDeinit ( FLEXIO_I2C_Type * base )`

Calling this API Resets the FlexIO I2C master shifer and timer config, module can't work unless the FLEXIO\_I2C\_MasterInit is called.

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

#### 32.4.7.4 void FLEXIO\_I2C\_MasterGetDefaultConfig ( flexio\_i2c\_master\_config\_t \* *masterConfig* )

The configuration can be used directly for calling the [FLEXIO\\_I2C\\_MasterInit\(\)](#).

Example:

```
flexio_i2c_master_config_t config;
FLEXIO_I2C_MasterGetDefaultConfig(&config);
```

## Parameters

|                     |                                                                  |
|---------------------|------------------------------------------------------------------|
| <i>masterConfig</i> | Pointer to <a href="#">flexio_i2c_master_config_t</a> structure. |
|---------------------|------------------------------------------------------------------|

#### 32.4.7.5 static void FLEXIO\_I2C\_MasterEnable ( FLEXIO\_I2C\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

|               |                                                             |
|---------------|-------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.       |
| <i>enable</i> | Pass true to enable module, false does not have any effect. |

#### 32.4.7.6 uint32\_t FLEXIO\_I2C\_MasterGetStatusFlags ( FLEXIO\_I2C\_Type \* *base* )

## Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure |
|-------------|------------------------------------------------------|

## Returns

Status flag, use status flag to AND [\\_flexio\\_i2c\\_master\\_status\\_flags](#) can get the related status.

#### 32.4.7.7 void FLEXIO\_I2C\_MasterClearStatusFlags ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *mask* )



## Parameters

|             |                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                                                                                                                       |
| <i>mask</i> | Status flag. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kFLEXIO_I2C_RxFullFlag</li> <li>• kFLEXIO_I2C_ReceiveNakFlag</li> </ul> |

### 32.4.7.8 void FLEXIO\_I2C\_MasterEnableInterrupts ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                                                                                                                |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                                                                                          |
| <i>mask</i> | Interrupt source. Currently only one interrupt request source: <ul style="list-style-type: none"> <li>• kFLEXIO_I2C_TransferCompleteInterruptEnable</li> </ul> |

### 32.4.7.9 void FLEXIO\_I2C\_MasterDisableInterrupts ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>mask</i> | Interrupt source.                                     |

### 32.4.7.10 void FLEXIO\_I2C\_MasterSetBaudRate ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *baudRate\_Bps*, uint32\_t *srcClock\_Hz* )

## Parameters

|                     |                                                      |
|---------------------|------------------------------------------------------|
| <i>base</i>         | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure |
| <i>baudRate_Bps</i> | the baud rate value in HZ                            |

|                    |                    |
|--------------------|--------------------|
| <i>srcClock_Hz</i> | source clock in HZ |
|--------------------|--------------------|

**32.4.7.11 void FLEXIO\_I2C\_MasterStart ( FLEXIO\_I2C\_Type \* *base*, uint8\_t *address*, flexio\_i2c\_direction\_t *direction* )**

Note

This API should be called when the transfer configuration is ready to send a START signal and 7-bit address to the bus. This is a non-blocking API, which returns directly after the address is put into the data register but the address transfer is not finished on the bus. Ensure that the kFLEXIO\_I2C\_RxFullFlag status is asserted before calling this API.

Parameters

|                  |                                                                                                                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                                                                                                                                   |
| <i>address</i>   | 7-bit address.                                                                                                                                                                                          |
| <i>direction</i> | transfer direction. This parameter is one of the values in flexio_i2c_direction_t: <ul style="list-style-type: none"> <li>• kFLEXIO_I2C_Write: Transmit</li> <li>• kFLEXIO_I2C_Read: Receive</li> </ul> |

**32.4.7.12 void FLEXIO\_I2C\_MasterStop ( FLEXIO\_I2C\_Type \* *base* )**

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

**32.4.7.13 void FLEXIO\_I2C\_MasterRepeatedStart ( FLEXIO\_I2C\_Type \* *base* )**

Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

**32.4.7.14 void FLEXIO\_I2C\_MasterAbortStop ( FLEXIO\_I2C\_Type \* *base* )**

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

**32.4.7.15 void FLEXIO\_I2C\_MasterEnableAck ( FLEXIO\_I2C\_Type \* *base*, bool *enable* )**

## Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.    |
| <i>enable</i> | True to configure send ACK, false configure to send NAK. |

**32.4.7.16 status\_t FLEXIO\_I2C\_MasterSetTransferCount ( FLEXIO\_I2C\_Type \* *base*, uint16\_t *count* )**

## Note

Call this API before a transfer begins because the timer generates a number of clocks according to the number of bytes that need to be transferred.

## Parameters

|              |                                                                                      |
|--------------|--------------------------------------------------------------------------------------|
| <i>base</i>  | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                                |
| <i>count</i> | Number of bytes need to be transferred from a start signal to a re-start/stop signal |

## Return values

|                                |                                    |
|--------------------------------|------------------------------------|
| <i>kStatus_Success</i>         | Successfully configured the count. |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.         |

**32.4.7.17 static void FLEXIO\_I2C\_MasterWriteByte ( FLEXIO\_I2C\_Type \* *base*, uint32\_t *data* ) [inline], [static]**

## Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>data</i> | a byte of data.                                       |

### 32.4.7.18 static uint8\_t FLEXIO\_I2C\_MasterReadByte ( FLEXIO\_I2C\_Type \* *base* ) [inline], [static]

## Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the data is ready in the register.

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
|-------------|-------------------------------------------------------|

## Returns

data byte read.

### 32.4.7.19 status\_t FLEXIO\_I2C\_MasterWriteBlocking ( FLEXIO\_I2C\_Type \* *base*, const uint8\_t \* *txBuff*, uint8\_t *txSize* )

## Note

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

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>txBuff</i> | The data bytes to send.                               |
| <i>txSize</i> | The number of data bytes to send.                     |

## Return values

|                                         |                                  |
|-----------------------------------------|----------------------------------|
| <i>kStatus_Success</i>                  | Successfully write data.         |
| <i>kStatus_FLEXIO_I2C_-<br/>Nak</i>     | Receive NAK during writing data. |
| <i>kStatus_FLEXIO_I2C_-<br/>Timeout</i> | Timeout polling status flags.    |

### 32.4.7.20 `status_t FLEXIO_I2C_MasterReadBlocking ( FLEXIO_I2C_Type * base, uint8_t * rxBuff, uint8_t rxSize )`

Note

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

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure. |
| <i>rxBuff</i> | The buffer to store the received bytes.               |
| <i>rxSize</i> | The number of data bytes to be received.              |

Return values

|                                         |                               |
|-----------------------------------------|-------------------------------|
| <i>kStatus_Success</i>                  | Successfully read data.       |
| <i>kStatus_FLEXIO_I2C_-<br/>Timeout</i> | Timeout polling status flags. |

### 32.4.7.21 `status_t FLEXIO_I2C_MasterTransferBlocking ( FLEXIO_I2C_Type * base, flexio_i2c_master_transfer_t * xfer )`

Note

The API does not return until the transfer succeeds or fails due to receiving NAK.

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2C_Type</a> structure.              |
| <i>xfer</i> | pointer to <a href="#">flexio_i2c_master_transfer_t</a> structure. |

Returns

status of `status_t`.

### 32.4.7.22 `status_t FLEXIO_I2C_MasterTransferCreateHandle ( FLEXIO_I2C_Type * base, flexio_i2c_master_handle_t * handle, flexio_i2c_master_transfer_callback_t callback, void * userData )`

## Parameters

|                 |                                                                              |
|-----------------|------------------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                        |
| <i>handle</i>   | Pointer to flexio_i2c_master_handle_t structure to store the transfer state. |
| <i>callback</i> | Pointer to user callback function.                                           |
| <i>userData</i> | User param passed to the callback function.                                  |

## Return values

|                           |                                                |
|---------------------------|------------------------------------------------|
| <i>kStatus_Success</i>    | Successfully create the handle.                |
| <i>kStatus_OutOfRange</i> | The FlexIO type/handle/isr table out of range. |

### 32.4.7.23 status\_t FLEXIO\_I2C\_MasterTransferNonBlocking ( FLEXIO\_I2C\_Type \* *base*, flexio\_i2c\_master\_handle\_t \* *handle*, flexio\_i2c\_master\_transfer\_t \* *xfer* )

## Note

The API returns immediately after the transfer initiates. Call FLEXIO\_I2C\_MasterTransferGetCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus\_FLEXIO\_I2C\_Busy, the transfer is finished.

## Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure                            |
| <i>handle</i> | Pointer to flexio_i2c_master_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">flexio_i2c_master_transfer_t</a> structure               |

## Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                       |
| <i>kStatus_FLEXIO_I2C_Busy</i> | FlexIO I2C is not idle, is running another transfer. |

### 32.4.7.24 status\_t FLEXIO\_I2C\_MasterTransferGetCount ( FLEXIO\_I2C\_Type \* *base*, flexio\_i2c\_master\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure.                            |
| <i>handle</i> | Pointer to flexio_i2c_master_handle_t structure which stores the transfer state. |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction.              |

## Return values

|                                     |                                                                |
|-------------------------------------|----------------------------------------------------------------|
| <i>kStatus_InvalidArgument</i>      | count is Invalid.                                              |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |
| <i>kStatus_Success</i>              | Successfully return the count.                                 |

### 32.4.7.25 void FLEXIO\_I2C\_MasterTransferAbort ( FLEXIO\_I2C\_Type \* *base*, flexio\_i2c\_master\_handle\_t \* *handle* )

## Note

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

## Parameters

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure                            |
| <i>handle</i> | Pointer to flexio_i2c_master_handle_t structure which stores the transfer state |

### 32.4.7.26 void FLEXIO\_I2C\_MasterTransferHandleIRQ ( void \* *i2cType*, void \* *i2cHandle* )

## Parameters

|                  |                                                                   |
|------------------|-------------------------------------------------------------------|
| <i>i2cType</i>   | Pointer to <a href="#">FLEXIO_I2C_Type</a> structure              |
| <i>i2cHandle</i> | Pointer to <a href="#">flexio_i2c_master_transfer_t</a> structure |

## 32.5 FlexIO I2S Driver

### 32.5.1 Overview

The MCUXpresso SDK provides a peripheral driver for I2S function using Flexible I/O module of MCU-Xpresso SDK devices.

The FlexIO I2S driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs.

Functional APIs can be used for FlexIO I2S initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO I2S peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. FlexIO I2S functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `sai_handle_t` as the first parameter. Initialize the handle by calling the `FlexIO_I2S_TransferTxCreateHandle()` or `FlexIO_I2S_TransferRxCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO\\_I2S\\_TransferSendNonBlocking\(\)](#) and [FLEXIO\\_I2S\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_I2S_TxIdle` and `kStatus_FLEXIO_I2S_RxIdle` status.

### 32.5.2 Typical use case

#### 32.5.2.1 FlexIO I2S send/receive using an interrupt method

```
sai_handle_t g_saiTxHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
volatile bool rxFinished;
const uint8_t sendData[] = [.....];

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
 userData = userData;

 if (kStatus_FLEXIO_I2S_TxIdle == status)
 {
 txFinished = true;
 }
}

void main(void)
{
 //...

 FLEXIO_I2S_TxGetDefaultConfig(&user_config);
```



```

FLEXIO_I2S_TxInit(FLEXIO_I2S0, &user_config);
FLEXIO_I2S_TransferTxCreateHandle(FLEXIO_I2S0, &g_saiHandle,
 FLEXIO_I2S_UserCallback, NULL);

//Configures the SAI format.
FLEXIO_I2S_TransferTxSetTransferFormat(FLEXIO_I2S0, &g_saiHandle, mclkSource, mclk);

// Prepares to send.
sendXfer.data = sendData
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendNonBlocking(FLEXIO_I2S0, &g_saiHandle, &
 sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}

```

### 32.5.2.2 FLEXIO\_I2S send/receive using a DMA method

```

sai_handle_t g_saiHandle;
dma_handle_t g_saiTxDmaHandle;
dma_handle_t g_saiRxDmaHandle;
sai_config_t user_config;
sai_transfer_t sendXfer;
volatile bool txFinished;
uint8_t sendData[] = ...;

void FLEXIO_I2S_UserCallback(sai_handle_t *handle, status_t status, void *userData)
{
 userData = userData;

 if (kStatus_FLEXIO_I2S_TxIdle == status)
 {
 txFinished = true;
 }
}

void main(void)
{
 //...

 FLEXIO_I2S_TxGetDefaultConfig(&user_config);
 FLEXIO_I2S_TxInit(FLEXIO_I2S0, &user_config);

 // Sets up the DMA.
 DMAMUX_Init(DMAMUX0);
 DMAMUX_SetSource(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL, FLEXIO_I2S_TX_DMA_REQUEST);
 DMAMUX_EnableChannel(DMAMUX0, FLEXIO_I2S_TX_DMA_CHANNEL);

 DMA_Init(DMA0);

 /* Creates the DMA handle. */
 DMA_TransferTxCreateHandle(&g_saiTxDmaHandle, DMA0, FLEXIO_I2S_TX_DMA_CHANNEL);

 FLEXIO_I2S_TransferTxCreateHandleDMA(FLEXIO_I2S0, &g_saiTxDmaHandle, FLEXIO_I2S_UserCallback, NULL);

 // Prepares to send.
 sendXfer.data = sendData

```

```

sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_I2S_TransferSendDMA(&g_saiHandle, &sendXfer);

// Waiting to send is finished.
while (!txFinished)
{
}

// ...
}

```

## Modules

- [FlexIO eDMA I2S Driver](#)

## Data Structures

- struct [FLEXIO\\_I2S\\_Type](#)  
*Define FlexIO I2S access structure typedef. [More...](#)*
- struct [flexio\\_i2s\\_config\\_t](#)  
*FlexIO I2S configure structure. [More...](#)*
- struct [flexio\\_i2s\\_format\\_t](#)  
*FlexIO I2S audio format, FlexIO I2S only support the same format in Tx and Rx. [More...](#)*
- struct [flexio\\_i2s\\_transfer\\_t](#)  
*Define FlexIO I2S transfer structure. [More...](#)*
- struct [flexio\\_i2s\\_handle\\_t](#)  
*Define FlexIO I2S handle structure. [More...](#)*

## Macros

- #define [I2S\\_RETRY\\_TIMES](#) 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/  
*Retry times for waiting flag.*
- #define [FLEXIO\\_I2S\\_XFER\\_QUEUE\\_SIZE](#) (4U)  
*FlexIO I2S transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* [flexio\\_i2s\\_callback\\_t](#) )(FLEXIO\_I2S\_Type \*base, flexio\_i2s\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*FlexIO I2S xfer callback prototype.*

## Enumerations

- enum {  
`kStatus_FLEXIO_I2S_Idle` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 0),  
`kStatus_FLEXIO_I2S_TxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 1),  
`kStatus_FLEXIO_I2S_RxBusy` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 2),  
`kStatus_FLEXIO_I2S_Error` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 3),  
`kStatus_FLEXIO_I2S_QueueFull` = MAKE\_STATUS(kStatusGroup\_FLEXIO\_I2S, 4),  
`kStatus_FLEXIO_I2S_Timeout` }  
*FlexIO I2S transfer status.*
- enum `flexio_i2s_master_slave_t` {  
`kFLEXIO_I2S_Master` = 0x0U,  
`kFLEXIO_I2S_Slave` = 0x1U }  
*Master or slave mode.*
- enum {  
`kFLEXIO_I2S_TxDataRegEmptyInterruptEnable` = 0x1U,  
`kFLEXIO_I2S_RxDataRegFullInterruptEnable` = 0x2U }  
*\_flexio\_i2s\_interrupt\_enable Define FlexIO FlexIO I2S interrupt mask.*
- enum {  
`kFLEXIO_I2S_TxDataRegEmptyFlag` = 0x1U,  
`kFLEXIO_I2S_RxDataRegFullFlag` = 0x2U }  
*\_flexio\_i2s\_status\_flags Define FlexIO FlexIO I2S status mask.*
- enum `flexio_i2s_sample_rate_t` {  
`kFLEXIO_I2S_SampleRate8KHz` = 8000U,  
`kFLEXIO_I2S_SampleRate11025Hz` = 11025U,  
`kFLEXIO_I2S_SampleRate12KHz` = 12000U,  
`kFLEXIO_I2S_SampleRate16KHz` = 16000U,  
`kFLEXIO_I2S_SampleRate22050Hz` = 22050U,  
`kFLEXIO_I2S_SampleRate24KHz` = 24000U,  
`kFLEXIO_I2S_SampleRate32KHz` = 32000U,  
`kFLEXIO_I2S_SampleRate44100Hz` = 44100U,  
`kFLEXIO_I2S_SampleRate48KHz` = 48000U,  
`kFLEXIO_I2S_SampleRate96KHz` = 96000U }  
*Audio sample rate.*
- enum `flexio_i2s_word_width_t` {  
`kFLEXIO_I2S_WordWidth8bits` = 8U,  
`kFLEXIO_I2S_WordWidth16bits` = 16U,  
`kFLEXIO_I2S_WordWidth24bits` = 24U,  
`kFLEXIO_I2S_WordWidth32bits` = 32U }  
*Audio word width.*

## Driver version

- #define `FSL_FLEXIO_I2S_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 0))  
*FlexIO I2S driver version 2.2.0.*

## Initialization and deinitialization

- void [FLEXIO\\_I2S\\_Init](#) ([FLEXIO\\_I2S\\_Type](#) \*base, const [flexio\\_i2s\\_config\\_t](#) \*config)  
*Initializes the FlexIO I2S.*
- void [FLEXIO\\_I2S\\_GetDefaultConfig](#) ([flexio\\_i2s\\_config\\_t](#) \*config)  
*Sets the FlexIO I2S configuration structure to default values.*
- void [FLEXIO\\_I2S\\_Deinit](#) ([FLEXIO\\_I2S\\_Type](#) \*base)  
*De-initializes the FlexIO I2S.*
- static void [FLEXIO\\_I2S\\_Enable](#) ([FLEXIO\\_I2S\\_Type](#) \*base, bool enable)  
*Enables/disables the FlexIO I2S module operation.*

## Status

- uint32\_t [FLEXIO\\_I2S\\_GetStatusFlags](#) ([FLEXIO\\_I2S\\_Type](#) \*base)  
*Gets the FlexIO I2S status flags.*

## Interrupts

- void [FLEXIO\\_I2S\\_EnableInterrupts](#) ([FLEXIO\\_I2S\\_Type](#) \*base, uint32\_t mask)  
*Enables the FlexIO I2S interrupt.*
- void [FLEXIO\\_I2S\\_DisableInterrupts](#) ([FLEXIO\\_I2S\\_Type](#) \*base, uint32\_t mask)  
*Disables the FlexIO I2S interrupt.*

## DMA Control

- static void [FLEXIO\\_I2S\\_TxEnableDMA](#) ([FLEXIO\\_I2S\\_Type](#) \*base, bool enable)  
*Enables/disables the FlexIO I2S Tx DMA requests.*
- static void [FLEXIO\\_I2S\\_RxEnableDMA](#) ([FLEXIO\\_I2S\\_Type](#) \*base, bool enable)  
*Enables/disables the FlexIO I2S Rx DMA requests.*
- static uint32\_t [FLEXIO\\_I2S\\_TxGetDataRegisterAddress](#) ([FLEXIO\\_I2S\\_Type](#) \*base)  
*Gets the FlexIO I2S send data register address.*
- static uint32\_t [FLEXIO\\_I2S\\_RxGetDataRegisterAddress](#) ([FLEXIO\\_I2S\\_Type](#) \*base)  
*Gets the FlexIO I2S receive data register address.*

## Bus Operations

- void [FLEXIO\\_I2S\\_MasterSetFormat](#) ([FLEXIO\\_I2S\\_Type](#) \*base, [flexio\\_i2s\\_format\\_t](#) \*format, uint32\_t srcClock\_Hz)  
*Configures the FlexIO I2S audio format in master mode.*
- void [FLEXIO\\_I2S\\_SlaveSetFormat](#) ([FLEXIO\\_I2S\\_Type](#) \*base, [flexio\\_i2s\\_format\\_t](#) \*format)  
*Configures the FlexIO I2S audio format in slave mode.*
- [status\\_t](#) [FLEXIO\\_I2S\\_WriteBlocking](#) ([FLEXIO\\_I2S\\_Type](#) \*base, uint8\_t bitWidth, uint8\_t \*tx-Data, size\_t size)  
*Sends data using a blocking method.*
- static void [FLEXIO\\_I2S\\_WriteData](#) ([FLEXIO\\_I2S\\_Type](#) \*base, uint8\_t bitWidth, uint32\_t data)

- Writes data into a data register.
- `status_t FLEXIO_I2S_ReadBlocking (FLEXIO_I2S_Type *base, uint8_t bitWidth, uint8_t *rxData, size_t size)`  
Receives a piece of data using a blocking method.
- `static uint32_t FLEXIO_I2S_ReadData (FLEXIO_I2S_Type *base)`  
Reads a data from the data register.

## Transactional

- `void FLEXIO_I2S_TransferTxCreateHandle (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)`  
Initializes the FlexIO I2S handle.
- `void FLEXIO_I2S_TransferSetFormat (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_format_t *format, uint32_t srcClock_Hz)`  
Configures the FlexIO I2S audio format.
- `void FLEXIO_I2S_TransferRxCreateHandle (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_callback_t callback, void *userData)`  
Initializes the FlexIO I2S receive handle.
- `status_t FLEXIO_I2S_TransferSendNonBlocking (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_transfer_t *xfer)`  
Performs an interrupt non-blocking send transfer on FlexIO I2S.
- `status_t FLEXIO_I2S_TransferReceiveNonBlocking (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, flexio_i2s_transfer_t *xfer)`  
Performs an interrupt non-blocking receive transfer on FlexIO I2S.
- `void FLEXIO_I2S_TransferAbortSend (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)`  
Aborts the current send.
- `void FLEXIO_I2S_TransferAbortReceive (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle)`  
Aborts the current receive.
- `status_t FLEXIO_I2S_TransferGetSendCount (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`  
Gets the remaining bytes to be sent.
- `status_t FLEXIO_I2S_TransferGetReceiveCount (FLEXIO_I2S_Type *base, flexio_i2s_handle_t *handle, size_t *count)`  
Gets the remaining bytes to be received.
- `void FLEXIO_I2S_TransferTxHandleIRQ (void *i2sBase, void *i2sHandle)`  
Tx interrupt handler.
- `void FLEXIO_I2S_TransferRxHandleIRQ (void *i2sBase, void *i2sHandle)`  
Rx interrupt handler.

## 32.5.3 Data Structure Documentation

### 32.5.3.1 struct FLEXIO\_I2S\_Type

#### Data Fields

- `FLEXIO_Type * flexioBase`  
FlexIO base pointer.

- uint8\_t [txPinIndex](#)  
*Tx data pin index in FlexIO pins.*
- uint8\_t [rxPinIndex](#)  
*Rx data pin index.*
- uint8\_t [bclkPinIndex](#)  
*Bit clock pin index.*
- uint8\_t [fsPinIndex](#)  
*Frame sync pin index.*
- uint8\_t [txShifterIndex](#)  
*Tx data shifter index.*
- uint8\_t [rxShifterIndex](#)  
*Rx data shifter index.*
- uint8\_t [bclkTimerIndex](#)  
*Bit clock timer index.*
- uint8\_t [fsTimerIndex](#)  
*Frame sync timer index.*

### 32.5.3.2 struct flexio\_i2s\_config\_t

#### Data Fields

- bool [enableI2S](#)  
*Enable FlexIO I2S.*
- [flexio\\_i2s\\_master\\_slave\\_t](#) masterSlave  
*Master or slave.*
- [flexio\\_pin\\_polarity\\_t](#) txPinPolarity  
*Tx data pin polarity, active high or low.*
- [flexio\\_pin\\_polarity\\_t](#) rxPinPolarity  
*Rx data pin polarity.*
- [flexio\\_pin\\_polarity\\_t](#) bclkPinPolarity  
*Bit clock pin polarity.*
- [flexio\\_pin\\_polarity\\_t](#) fsPinPolarity  
*Frame sync pin polarity.*
- [flexio\\_shifter\\_timer\\_polarity\\_t](#) txTimerPolarity  
*Tx data valid on bclk rising or falling edge.*
- [flexio\\_shifter\\_timer\\_polarity\\_t](#) rxTimerPolarity  
*Rx data valid on bclk rising or falling edge.*

### 32.5.3.3 struct flexio\_i2s\_format\_t

#### Data Fields

- uint8\_t [bitWidth](#)  
*Bit width of audio data, always 8/16/24/32 bits.*
- uint32\_t [sampleRate\\_Hz](#)  
*Sample rate of the audio data.*

### 32.5.3.4 struct flexio\_i2s\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)  
*Data buffer start pointer.*
- size\_t [dataSize](#)  
*Bytes to be transferred.*

#### Field Documentation

(1) size\_t flexio\_i2s\_transfer\_t::dataSize

### 32.5.3.5 struct \_flexio\_i2s\_handle

#### Data Fields

- uint32\_t [state](#)  
*Internal state.*
- [flexio\\_i2s\\_callback\\_t](#) [callback](#)  
*Callback function called at transfer event.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*
- uint8\_t [bitWidth](#)  
*Bit width for transfer, 8/16/24/32bits.*
- [flexio\\_i2s\\_transfer\\_t](#) [queue](#) [FLEXIO\_I2S\_XFER\_QUEUE\_SIZE]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [FLEXIO\_I2S\_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.*

## 32.5.4 Macro Definition Documentation

32.5.4.1 #define FSL\_FLEXIO\_I2S\_DRIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

32.5.4.2 #define I2S\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/

32.5.4.3 #define FLEXIO\_I2S\_XFER\_QUEUE\_SIZE (4U)

## 32.5.5 Enumeration Type Documentation

### 32.5.5.1 anonymous enum

Enumerator

*kStatus\_FLEXIO\_I2S\_Idle* FlexIO I2S is in idle state.  
*kStatus\_FLEXIO\_I2S\_TxBusy* FlexIO I2S Tx is busy.  
*kStatus\_FLEXIO\_I2S\_RxBusy* FlexIO I2S Rx is busy.  
*kStatus\_FLEXIO\_I2S\_Error* FlexIO I2S error occurred.  
*kStatus\_FLEXIO\_I2S\_QueueFull* FlexIO I2S transfer queue is full.  
*kStatus\_FLEXIO\_I2S\_Timeout* FlexIO I2S timeout polling status flags.

### 32.5.5.2 enum flexio\_i2s\_master\_slave\_t

Enumerator

*kFLEXIO\_I2S\_Master* Master mode.  
*kFLEXIO\_I2S\_Slave* Slave mode.

### 32.5.5.3 anonymous enum

Enumerator

*kFLEXIO\_I2S\_TxDataRegEmptyInterruptEnable* Transmit buffer empty interrupt enable.  
*kFLEXIO\_I2S\_RxDataRegFullInterruptEnable* Receive buffer full interrupt enable.

### 32.5.5.4 anonymous enum

Enumerator

*kFLEXIO\_I2S\_TxDataRegEmptyFlag* Transmit buffer empty flag.  
*kFLEXIO\_I2S\_RxDataRegFullFlag* Receive buffer full flag.

### 32.5.5.5 enum flexio\_i2s\_sample\_rate\_t

Enumerator

*kFLEXIO\_I2S\_SampleRate8KHz* Sample rate 8000Hz.  
*kFLEXIO\_I2S\_SampleRate11025Hz* Sample rate 11025Hz.  
*kFLEXIO\_I2S\_SampleRate12KHz* Sample rate 12000Hz.  
*kFLEXIO\_I2S\_SampleRate16KHz* Sample rate 16000Hz.  
*kFLEXIO\_I2S\_SampleRate22050Hz* Sample rate 22050Hz.  
*kFLEXIO\_I2S\_SampleRate24KHz* Sample rate 24000Hz.



***kFLEXIO\_I2S\_SampleRate32KHz*** Sample rate 32000Hz.  
***kFLEXIO\_I2S\_SampleRate44100Hz*** Sample rate 44100Hz.  
***kFLEXIO\_I2S\_SampleRate48KHz*** Sample rate 48000Hz.  
***kFLEXIO\_I2S\_SampleRate96KHz*** Sample rate 96000Hz.

### 32.5.5.6 enum flexio\_i2s\_word\_width\_t

Enumerator

***kFLEXIO\_I2S\_WordWidth8bits*** Audio data width 8 bits.  
***kFLEXIO\_I2S\_WordWidth16bits*** Audio data width 16 bits.  
***kFLEXIO\_I2S\_WordWidth24bits*** Audio data width 24 bits.  
***kFLEXIO\_I2S\_WordWidth32bits*** Audio data width 32 bits.

## 32.5.6 Function Documentation

### 32.5.6.1 void FLEXIO\_I2S\_Init ( FLEXIO\_I2S\_Type \* ***base***, const flexio\_i2s\_config\_t \* ***config*** )

This API configures FlexIO pins and shifter to I2S and configures the FlexIO I2S with a configuration structure. The configuration structure can be filled by the user, or be set with default values by [FLEXIO\\_I2S\\_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the FlexIO I2S driver. Otherwise, any access to the FlexIO I2S module can cause hard fault because the clock is not enabled.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | FlexIO I2S base pointer         |
| <i>config</i> | FlexIO I2S configure structure. |

### 32.5.6.2 void FLEXIO\_I2S\_GetDefaultConfig ( flexio\_i2s\_config\_t \* ***config*** )

The purpose of this API is to get the configuration structure initialized for use in [FLEXIO\\_I2S\\_Init\(\)](#). Users may use the initialized structure unchanged in [FLEXIO\\_I2S\\_Init\(\)](#) or modify some fields of the structure before calling [FLEXIO\\_I2S\\_Init\(\)](#).

## Parameters

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

**32.5.6.3 void FLEXIO\_I2S\_Deinit ( FLEXIO\_I2S\_Type \* *base* )**

Calling this API resets the FlexIO I2S shifter and timer config. After calling this API, call the FLEXIO\_I2S\_Init to use the FlexIO I2S module.

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | FlexIO I2S base pointer |
|-------------|-------------------------|

**32.5.6.4 static void FLEXIO\_I2S\_Enable ( FLEXIO\_I2S\_Type \* *base*, bool *enable* )  
[inline], [static]**

## Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a>      |
| <i>enable</i> | True to enable, false dose not have any effect. |

**32.5.6.5 uint32\_t FLEXIO\_I2S\_GetStatusFlags ( FLEXIO\_I2S\_Type \* *base* )**

## Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

## Returns

Status flag, which are ORed by the enumerators in the `_flexio_i2s_status_flags`.

**32.5.6.6 void FLEXIO\_I2S\_EnableInterrupts ( FLEXIO\_I2S\_Type \* *base*, uint32\_t *mask* )**

This function enables the FlexIO UART interrupt.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>mask</i> | interrupt source                                     |

**32.5.6.7 void FLEXIO\_I2S\_DisableInterrupts ( FLEXIO\_I2S\_Type \* *base*, uint32\_t *mask* )**

This function enables the FlexIO UART interrupt.

Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>mask</i> | interrupt source                                     |

**32.5.6.8 static void FLEXIO\_I2S\_TxEnableDMA ( FLEXIO\_I2S\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FlexIO I2S base pointer                         |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

**32.5.6.9 static void FLEXIO\_I2S\_RxEnableDMA ( FLEXIO\_I2S\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>base</i>   | FlexIO I2S base pointer                         |
| <i>enable</i> | True means enable DMA, false means disable DMA. |

**32.5.6.10 static uint32\_t FLEXIO\_I2S\_TxGetDataRegisterAddress ( FLEXIO\_I2S\_Type \* *base* ) [inline], [static]**

This function returns the I2S data register address, mainly used by DMA/eDMA.

## Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

## Returns

FlexIO i2s send data register address.

**32.5.6.11** `static uint32_t FLEXIO_I2S_RxGetDataRegisterAddress ( FLEXIO_I2S_Type * base ) [inline], [static]`

This function returns the I2S data register address, mainly used by DMA/eDMA.

## Parameters

|             |                                                      |
|-------------|------------------------------------------------------|
| <i>base</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
|-------------|------------------------------------------------------|

## Returns

FlexIO i2s receive data register address.

**32.5.6.12** `void FLEXIO_I2S_MasterSetFormat ( FLEXIO_I2S_Type * base, flexio_i2s_format_t * format, uint32_t srcClock_Hz )`

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

## Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>base</i>        | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>format</i>      | Pointer to FlexIO I2S audio data format structure.   |
| <i>srcClock_Hz</i> | I2S master clock source frequency in Hz.             |

**32.5.6.13** `void FLEXIO_I2S_SlaveSetFormat ( FLEXIO_I2S_Type * base, flexio_i2s_format_t * format )`

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

## Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure |
| <i>format</i> | Pointer to FlexIO I2S audio data format structure.   |

#### 32.5.6.14 **status\_t FLEXIO\_I2S\_WriteBlocking ( FLEXIO\_I2S\_Type \* *base*, uint8\_t *bitWidth*, uint8\_t \* *txData*, size\_t *size* )**

## Note

This function blocks via polling until data is ready to be sent.

## Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer.                                |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>txData</i>   | Pointer to the data to be written.                      |
| <i>size</i>     | Bytes to be written.                                    |

## Return values

|                                   |                               |
|-----------------------------------|-------------------------------|
| <i>kStatus_Success</i>            | Successfully write data.      |
| <i>kStatus_FLEXIO_I2C_Timeout</i> | Timeout polling status flags. |

#### 32.5.6.15 **static void FLEXIO\_I2S\_WriteData ( FLEXIO\_I2S\_Type \* *base*, uint8\_t *bitWidth*, uint32\_t *data* ) [inline], [static]**

## Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer.                                |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>data</i>     | Data to be written.                                     |

#### 32.5.6.16 **status\_t FLEXIO\_I2S\_ReadBlocking ( FLEXIO\_I2S\_Type \* *base*, uint8\_t *bitWidth*, uint8\_t \* *rxData*, size\_t *size* )**

## Note

This function blocks via polling until data is ready to be sent.

## Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>base</i>     | FlexIO I2S base pointer                                 |
| <i>bitWidth</i> | How many bits in a audio word, usually 8/16/24/32 bits. |
| <i>rxData</i>   | Pointer to the data to be read.                         |
| <i>size</i>     | Bytes to be read.                                       |

## Return values

|                                   |                               |
|-----------------------------------|-------------------------------|
| <i>kStatus_Success</i>            | Successfully read data.       |
| <i>kStatus_FLEXIO_I2C_Timeout</i> | Timeout polling status flags. |

### 32.5.6.17 static uint32\_t FLEXIO\_I2S\_ReadData ( FLEXIO\_I2S\_Type \* *base* ) [inline], [static]

## Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | FlexIO I2S base pointer |
|-------------|-------------------------|

## Returns

Data read from data register.

### 32.5.6.18 void FLEXIO\_I2S\_TransferTxCreateHandle ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_callback\_t *callback*, void \* *userData* )

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

## Parameters

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure                  |
| <i>handle</i>   | Pointer to flexio_i2s_handle_t structure to store the transfer state. |
| <i>callback</i> | FlexIO I2S callback function, which is called while finished a block. |
| <i>userData</i> | User parameter for the FlexIO I2S callback.                           |

**32.5.6.19 void FLEXIO\_I2S\_TransferSetFormat ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed at run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred.

Parameters

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>        | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                                        |
| <i>handle</i>      | FlexIO I2S handle pointer.                                                                   |
| <i>format</i>      | Pointer to audio data format structure.                                                      |
| <i>srcClock_Hz</i> | FlexIO I2S bit clock source frequency in Hz. This parameter should be 0 while in slave mode. |

**32.5.6.20 void FLEXIO\_I2S\_TransferRxCreateHandle ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_callback\_t *callback*, void \* *userData* )**

This function initializes the FlexIO I2S handle which can be used for other FlexIO I2S transactional APIs. Call this API once to get the initialized handle.

Parameters

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| <i>base</i>     | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                 |
| <i>handle</i>   | Pointer to flexio_i2s_handle_t structure to store the transfer state. |
| <i>callback</i> | FlexIO I2S callback function, which is called while finished a block. |
| <i>userData</i> | User parameter for the FlexIO I2S callback.                           |

**32.5.6.21 status\_t FLEXIO\_I2S\_TransferSendNonBlocking ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )**

## Note

The API returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetRemainingBytes to poll the transfer status and check whether the transfer is finished. If the return status is 0, the transfer is finished.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| <i>xfer</i>   | Pointer to <a href="#">flexio_i2s_transfer_t</a> structure               |

## Return values

|                                   |                                                                                    |
|-----------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>            | Successfully start the data transmission.                                          |
| <i>kStatus_FLEXIO_I2S_Tx-Busy</i> | Previous transmission still not finished, data not all written to TX register yet. |
| <i>kStatus_InvalidArgument</i>    | The input parameter is invalid.                                                    |

### 32.5.6.22 status\_t FLEXIO\_I2S\_TransferReceiveNonBlocking ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )

## Note

The API returns immediately after transfer initiates. Call FLEXIO\_I2S\_GetRemainingBytes to poll the transfer status to check whether the transfer is finished. If the return status is 0, the transfer is finished.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| <i>xfer</i>   | Pointer to <a href="#">flexio_i2s_transfer_t</a> structure               |

## Return values

|                        |                                      |
|------------------------|--------------------------------------|
| <i>kStatus_Success</i> | Successfully start the data receive. |
|------------------------|--------------------------------------|



|                                  |                                      |
|----------------------------------|--------------------------------------|
| <i>kStatus_FLEXIO_I2S_RxBusy</i> | Previous receive still not finished. |
| <i>kStatus_InvalidArgument</i>   | The input parameter is invalid.      |

### 32.5.6.23 void FLEXIO\_I2S\_TransferAbortSend ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle* )

#### Note

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

#### Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |

### 32.5.6.24 void FLEXIO\_I2S\_TransferAbortReceive ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle* )

#### Note

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

#### Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |

### 32.5.6.25 status\_t FLEXIO\_I2S\_TransferGetSendCount ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, size\_t \* *count* )

#### Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| <i>count</i>  | Bytes sent.                                                              |

Return values

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

### 32.5.6.26 status\_t FLEXIO\_I2S\_TransferGetReceiveCount ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure.                    |
| <i>handle</i> | Pointer to flexio_i2s_handle_t structure which stores the transfer state |
| <i>count</i>  | Bytes recieved.                                                          |

Returns

count Bytes received.

Return values

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

### 32.5.6.27 void FLEXIO\_I2S\_TransferTxHandleIRQ ( void \* *i2sBase*, void \* *i2sHandle* )

Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>i2sBase</i> | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure. |
|----------------|-------------------------------------------------------|

|                  |                                          |
|------------------|------------------------------------------|
| <i>i2sHandle</i> | Pointer to flexio_i2s_handle_t structure |
|------------------|------------------------------------------|

#### 32.5.6.28 void FLEXIO\_I2S\_TransferRxHandleIRQ ( void \* *i2sBase*, void \* *i2sHandle* )

Parameters

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <i>i2sBase</i>   | Pointer to <a href="#">FLEXIO_I2S_Type</a> structure. |
| <i>i2sHandle</i> | Pointer to flexio_i2s_handle_t structure.             |

## 32.5.7 FlexIO eDMA I2S Driver

### 32.5.7.1 Overview

#### Data Structures

- struct `flexio_i2s_edma_handle_t`  
*FlexIO I2S DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* `flexio_i2s_edma_callback_t`)(`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `status_t` status, void \*userData)  
*FlexIO I2S eDMA transfer callback function for finish and error.*

#### Driver version

- #define `FSL_FLEXIO_I2S_EDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 7))  
*FlexIO I2S EDMA driver version 2.1.7.*

#### eDMA Transactional

- void `FLEXIO_I2S_TransferTxCreateHandleEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_edma_callback_t` callback, void \*userData, `edma_handle_t` \*dmaHandle)  
*Initializes the FlexIO I2S eDMA handle.*
- void `FLEXIO_I2S_TransferRxCreateHandleEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_edma_callback_t` callback, void \*userData, `edma_handle_t` \*dmaHandle)  
*Initializes the FlexIO I2S Rx eDMA handle.*
- void `FLEXIO_I2S_TransferSetFormatEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_format_t` \*format, `uint32_t` srcClock\_Hz)  
*Configures the FlexIO I2S Tx audio format.*
- `status_t` `FLEXIO_I2S_TransferSendEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)  
*Performs a non-blocking FlexIO I2S transfer using DMA.*
- `status_t` `FLEXIO_I2S_TransferReceiveEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle, `flexio_i2s_transfer_t` \*xfer)  
*Performs a non-blocking FlexIO I2S receive using eDMA.*
- void `FLEXIO_I2S_TransferAbortSendEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle)  
*Aborts a FlexIO I2S transfer using eDMA.*
- void `FLEXIO_I2S_TransferAbortReceiveEDMA` (`FLEXIO_I2S_Type` \*base, `flexio_i2s_edma_handle_t` \*handle)  
*Aborts a FlexIO I2S receive using eDMA.*

- `status_t FLEXIO_I2S_TransferGetSendCountEDMA (FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t *handle, size_t *count)`  
*Gets the remaining bytes to be sent.*
- `status_t FLEXIO_I2S_TransferGetReceiveCountEDMA (FLEXIO_I2S_Type *base, flexio_i2s_edma_handle_t *handle, size_t *count)`  
*Get the remaining bytes to be received.*

## 32.5.7.2 Data Structure Documentation

### 32.5.7.2.1 struct flexio\_i2s\_edma\_handle

#### Data Fields

- `edma_handle_t * dmaHandle`  
*DMA handler for FlexIO I2S send.*
- `uint8_t bytesPerFrame`  
*Bytes in a frame.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint32_t state`  
*Internal state for FlexIO I2S eDMA transfer.*
- `flexio_i2s_edma_callback_t callback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`  
*User callback parameter.*
- `edma_tcd_t tcd [FLEXIO_I2S_XFER_QUEUE_SIZE+1U]`  
*TCD pool for eDMA transfer.*
- `flexio_i2s_transfer_t queue [FLEXIO_I2S_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*
- `size_t transferSize [FLEXIO_I2S_XFER_QUEUE_SIZE]`  
*Data bytes need to transfer.*
- `volatile uint8_t queueUser`  
*Index for user to queue transfer.*
- `volatile uint8_t queueDriver`  
*Index for driver to get the transfer data and size.*

#### Field Documentation

- (1) `uint8_t flexio_i2s_edma_handle_t::nbytes`
- (2) `edma_tcd_t flexio_i2s_edma_handle_t::tcd[FLEXIO_I2S_XFER_QUEUE_SIZE+1U]`
- (3) `flexio_i2s_transfer_t flexio_i2s_edma_handle_t::queue[FLEXIO_I2S_XFER_QUEUE_SIZE]`
- (4) `volatile uint8_t flexio_i2s_edma_handle_t::queueUser`

## 32.5.7.3 Macro Definition Documentation

### 32.5.7.3.1 #define FSL\_FLEXIO\_I2S\_EDMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 7))

### 32.5.7.4 Function Documentation

**32.5.7.4.1 void FLEXIO\_I2S\_TransferTxCreateHandleEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_edma\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *dmaHandle* )**

This function initializes the FlexIO I2S master DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

|                  |                                                                               |
|------------------|-------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                           |
| <i>handle</i>    | FlexIO I2S eDMA handle pointer.                                               |
| <i>callback</i>  | FlexIO I2S eDMA callback function called while finished a block.              |
| <i>userData</i>  | User parameter for callback.                                                  |
| <i>dmaHandle</i> | eDMA handle for FlexIO I2S. This handle is a static value allocated by users. |

**32.5.7.4.2 void FLEXIO\_I2S\_TransferRxCreateHandleEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_edma\_callback\_t *callback*, void \* *userData*, edma\_handle\_t \* *dmaHandle* )**

This function initializes the FlexIO I2S slave DMA handle which can be used for other FlexIO I2S master transactional APIs. Usually, for a specified FlexIO I2S instance, call this API once to get the initialized handle.

Parameters

|                  |                                                                               |
|------------------|-------------------------------------------------------------------------------|
| <i>base</i>      | FlexIO I2S peripheral base address.                                           |
| <i>handle</i>    | FlexIO I2S eDMA handle pointer.                                               |
| <i>callback</i>  | FlexIO I2S eDMA callback function called while finished a block.              |
| <i>userData</i>  | User parameter for callback.                                                  |
| <i>dmaHandle</i> | eDMA handle for FlexIO I2S. This handle is a static value allocated by users. |

**32.5.7.4.3 void FLEXIO\_I2S\_TransferSetFormatEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_format\_t \* *format*, uint32\_t *srcClock\_Hz* )**

Audio format can be changed in run-time of FlexIO I2S. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to format.

## Parameters

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <i>base</i>        | FlexIO I2S peripheral base address.                                          |
| <i>handle</i>      | FlexIO I2S eDMA handle pointer                                               |
| <i>format</i>      | Pointer to FlexIO I2S audio data format structure.                           |
| <i>srcClock_Hz</i> | FlexIO I2S clock source frequency in Hz, it should be 0 while in slave mode. |

#### 32.5.7.4.4 status\_t FLEXIO\_I2S\_TransferSendEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )

## Note

This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetTransferStatus to poll the transfer status and check whether the FlexIO I2S transfer is finished.

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

## Return values

|                                |                                            |
|--------------------------------|--------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S eDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.            |
| <i>kStatus_TxBusy</i>          | FlexIO I2S is busy sending data.           |

#### 32.5.7.4.5 status\_t FLEXIO\_I2S\_TransferReceiveEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, flexio\_i2s\_transfer\_t \* *xfer* )

## Note

This interface returned immediately after transfer initiates. Users should call FLEXIO\_I2S\_GetReceiveRemainingBytes to poll the transfer status and check whether the FlexIO I2S transfer is finished.

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>xfer</i>   | Pointer to DMA transfer structure.  |

## Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Start a FlexIO I2S eDMA receive successfully. |
| <i>kStatus_InvalidArgument</i> | The input arguments is invalid.               |
| <i>kStatus_RxBusy</i>          | FlexIO I2S is busy receiving data.            |

#### 32.5.7.4.6 void FLEXIO\_I2S\_TransferAbortSendEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle* )

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

#### 32.5.7.4.7 void FLEXIO\_I2S\_TransferAbortReceiveEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle* )

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |

#### 32.5.7.4.8 status\_t FLEXIO\_I2S\_TransferGetSendCountEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | FlexIO I2S peripheral base address. |
|-------------|-------------------------------------|



|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | FlexIO I2S DMA handle pointer. |
| <i>count</i>  | Bytes sent.                    |

Return values

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

#### 32.5.7.4.9 **status\_t FLEXIO\_I2S\_TransferGetReceiveCountEDMA ( FLEXIO\_I2S\_Type \* *base*, flexio\_i2s\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | FlexIO I2S peripheral base address. |
| <i>handle</i> | FlexIO I2S DMA handle pointer.      |
| <i>count</i>  | Bytes received.                     |

Return values

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

## 32.6 FlexIO MCU Interface LCD Driver

### 32.6.1 Overview

The MCUXpresso SDK provides a peripheral driver for LCD (8080 or 6800 interface) function using Flexible I/O module of MCUXpresso SDK devices.

The FlexIO LCD driver supports both 8-bit and 16-bit data bus, 8080 and 6800 interface. User could change the macro `FLEXIO_MCULCD_DATA_BUS_WIDTH` to choose 8-bit data bus or 16-bit data bus.

The FlexIO LCD driver supports three kinds of data transfer:

1. Send a data array. For example, send the LCD image data to the LCD controller.
2. Send a value many times. For example, send 0 many times to clean the LCD screen.
3. Read data into a data array. For example, read image from LCD controller.

The FlexIO LCD driver includes functional APIs and transactional APIs.

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

Transactional APIs are transaction target high level APIs. The transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code.

Transactional APIs support asynchronous transfer. This means that the function `FLEXIO_MCULCD_TransferNonBlocking` sets up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_FLEXIO_MCULCD_Idle` status.

### 32.6.2 Typical use case

#### 32.6.2.1 FlexIO LCD send/receive using functional APIs

This example shows how to send command, or write and read data using the functional APIs. The data bus is 16-bit.

```
uint16_t dataToSend[] = { ... };
uint16_t dataToReceive[] = { ... };

FLEXIO_MCULCD_Type flexioLcdDev;
flexio_MCULCD_transfer_t xfer;
flexio_MCULCD_config_t config;

FLEXIO_MCULCD_GetDefaultConfig(&config);
FLEXIO_MCULCD_Init(&flexioLcdDev, &config, 120000000);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
```

```

FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command1);
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command1;
xfer.dataCount = 0; // Only send command, no data transfer.
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteDataArrayBlocking(&flexioLcdDev, dataToSend, sizeof(
 dataToSend));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
xfer.dataCount = sizeof(dataToSend);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command2);
FLEXIO_MCULCD_WriteSameValueBlocking(&flexioLcdDev, value, 1000); //
 Send value 1000 times
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

// Method 1:
FLEXIO_MCULCD_StartTransfer(&flexioLcdDev);
FLEXIO_MCULCD_WriteCommandBlocking(&flexioLcdDev, command3);
FLEXIO_MCULCD_ReadDataArrayBlocking(&flexioLcdDev, dataToReceive, sizeof(
 dataToReceive));
FLEXIO_MCULCD_StopTransfer(&flexioLcdDev);

// Method 2:
xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
FLEXIO_MCULCD_TransferBlocking(&flexioLcdDev, &xfer);

```

### 32.6.2.2 FlexIO LCD send/receive using interrupt transactional APIs

```

flexio_MCULCD_handle_t handle;
volatile bool completeFlag = false;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_handle_t *handle,
 status_t status, void *userData)
{
 if (kStatus_FLEXIO_MCULCD_Idle == status)
 {
 completeFlag = true;
 }
}

void main(void)

```

```

{
 // Init the FlexIO LCD driver.
 FLEXIO_MCULCD_Init(...);

 // Create the transactional handle.
 FLEXIO_MCULCD_TransferCreateHandle(&flexioLcdDev, &handle,
 flexioLcdCallback, NULL);

 xfer.command = command1;
 xfer.dataCount = 0; // Only send command, no data transfer.
 completeFlag = false;
 FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &xfer);

 // When only send method, it is not necessary to wait for the callback,
 // because the command is sent using a blocking method internally. The
 // command has been sent out after the function FLEXIO_MCULCD_TransferNonBlocking
 // returns.
 while (!completeFlag)
 {
 }

 xfer.command = command2;
 xfer.mode = kFLEXIO_MCULCD_WriteArray;
 xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
 xfer.dataCount = sizeof(dataToSend);
 completeFlag = false;
 FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

 while (!completeFlag)
 {
 }

 xfer.command = command2;
 xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
 xfer.dataAddrOrSameValue = value;
 xfer.dataCount = 1000;
 completeFlag = false;
 FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

 while (!completeFlag)
 {
 }

 xfer.command = command3;
 xfer.mode = kFLEXIO_MCULCD_ReadArray;
 xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
 xfer.dataCount = sizeof(dataToReceive);
 completeFlag = false;
 FLEXIO_MCULCD_TransferNonBlocking(&flexioLcdDev, &handle, &xfer);

 while (!completeFlag)
 {
 }
}

```

## Modules

- [FlexIO eDMA MCU Interface LCD Driver](#)

*SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.*

## Data Structures

- struct [FLEXIO\\_MCULCD\\_Type](#)  
Define FlexIO MCULCD access structure typedef. [More...](#)
- struct [flexio\\_mculcd\\_config\\_t](#)  
Define FlexIO MCULCD configuration structure. [More...](#)
- struct [flexio\\_mculcd\\_transfer\\_t](#)  
Define FlexIO MCULCD transfer structure. [More...](#)
- struct [flexio\\_mculcd\\_handle\\_t](#)  
Define FlexIO MCULCD handle structure. [More...](#)

## Macros

- #define [FLEXIO\\_MCULCD\\_WAIT\\_COMPLETE\\_TIME](#) 512  
The delay time to wait for FLEXIO transmit complete.
- #define [FLEXIO\\_MCULCD\\_DATA\\_BUS\\_WIDTH](#) 16UL  
The data bus width, must be 8 or 16.

## Typedefs

- typedef void(\* [flexio\\_mculcd\\_pin\\_func\\_t](#))(bool set)  
Function to set or clear the CS and RS pin.
- typedef void(\* [flexio\\_mculcd\\_transfer\\_callback\\_t](#))([FLEXIO\\_MCULCD\\_Type](#) \*base, [flexio\\_mculcd\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
FlexIO MCULCD callback for finished transfer.

## Enumerations

- enum {  
  [kStatus\\_FLEXIO\\_MCULCD\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_MCULCD, 0),  
  [kStatus\\_FLEXIO\\_MCULCD\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_MCULCD, 1),  
  [kStatus\\_FLEXIO\\_MCULCD\\_Error](#) = MAKE\_STATUS(kStatusGroup\_FLEXIO\_MCULCD, 2) }  
FlexIO LCD transfer status.
- enum [flexio\\_mculcd\\_pixel\\_format\\_t](#) {  
  [kFLEXIO\\_MCULCD\\_RGB565](#) = 0,  
  [kFLEXIO\\_MCULCD\\_BGR565](#),  
  [kFLEXIO\\_MCULCD\\_RGB888](#),  
  [kFLEXIO\\_MCULCD\\_BGR888](#) }  
Define FlexIO MCULCD pixel format.
- enum [flexio\\_mculcd\\_bus\\_t](#) {  
  [kFLEXIO\\_MCULCD\\_8080](#),  
  [kFLEXIO\\_MCULCD\\_6800](#) }  
Define FlexIO MCULCD bus type.
- enum [\\_flexio\\_mculcd\\_interrupt\\_enable](#) {  
  [kFLEXIO\\_MCULCD\\_TxEmptyInterruptEnable](#) = (1U << 0U),

- ```
kFLEXIO_MCULCD_RxFullInterruptEnable = (1U << 1U) }
```
- Define FlexIO MCULCD interrupt mask.*
- enum `_flexio_mculcd_status_flags` {
`kFLEXIO_MCULCD_TxEmptyFlag` = (1U << 0U),
`kFLEXIO_MCULCD_RxFullFlag` = (1U << 1U) }
- Define FlexIO MCULCD status mask.*
- enum `_flexio_mculcd_dma_enable` {
`kFLEXIO_MCULCD_TxDmaEnable` = 0x1U,
`kFLEXIO_MCULCD_RxDmaEnable` = 0x2U }
- Define FlexIO MCULCD DMA mask.*
- enum `flexio_mculcd_transfer_mode_t` {
`kFLEXIO_MCULCD_ReadArray`,
`kFLEXIO_MCULCD_WriteArray`,
`kFLEXIO_MCULCD_WriteSameValue` }
- Transfer mode.*

Driver version

- #define `FSL_FLEXIO_MCULCD_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 6)`)
FlexIO MCULCD driver version.

FlexIO MCULCD Configuration

- `status_t` `FLEXIO_MCULCD_Init` (`FLEXIO_MCULCD_Type` *base, `flexio_mculcd_config_t` *config, `uint32_t` srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO MCULCD hardware, and configures the FlexIO MCULCD with FlexIO MCULCD configuration.
- void `FLEXIO_MCULCD_Deinit` (`FLEXIO_MCULCD_Type` *base)
Resets the FLEXIO_MCULCD timer and shifter configuration.
- void `FLEXIO_MCULCD_GetDefaultConfig` (`flexio_mculcd_config_t` *config)
Gets the default configuration to configure the FlexIO MCULCD.

Status

- `uint32_t` `FLEXIO_MCULCD_GetStatusFlags` (`FLEXIO_MCULCD_Type` *base)
Gets FlexIO MCULCD status flags.
- void `FLEXIO_MCULCD_ClearStatusFlags` (`FLEXIO_MCULCD_Type` *base, `uint32_t` mask)
Clears FlexIO MCULCD status flags.

Interrupts

- void `FLEXIO_MCULCD_EnableInterrupts` (`FLEXIO_MCULCD_Type` *base, `uint32_t` mask)
Enables the FlexIO MCULCD interrupt.
- void `FLEXIO_MCULCD_DisableInterrupts` (`FLEXIO_MCULCD_Type` *base, `uint32_t` mask)

Disables the FlexIO MCULCD interrupt.

DMA Control

- static void [FLEXIO_MCULCD_EnableTxDMA](#) ([FLEXIO_MCULCD_Type](#) *base, bool enable)
Enables/disables the FlexIO MCULCD transmit DMA.
- static void [FLEXIO_MCULCD_EnableRxDMA](#) ([FLEXIO_MCULCD_Type](#) *base, bool enable)
Enables/disables the FlexIO MCULCD receive DMA.
- static uint32_t [FLEXIO_MCULCD_GetTxDataRegisterAddress](#) ([FLEXIO_MCULCD_Type](#) *base)
Gets the FlexIO MCULCD transmit data register address.
- static uint32_t [FLEXIO_MCULCD_GetRxDataRegisterAddress](#) ([FLEXIO_MCULCD_Type](#) *base)
Gets the FlexIO MCULCD receive data register address.

Bus Operations

- [status_t FLEXIO_MCULCD_SetBaudRate](#) ([FLEXIO_MCULCD_Type](#) *base, uint32_t baudRate-_Bps, uint32_t srcClock_Hz)
Set desired baud rate.
- void [FLEXIO_MCULCD_SetSingleBeatWriteConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Configures the FLEXIO MCULCD to multiple beats write mode.
- void [FLEXIO_MCULCD_ClearSingleBeatWriteConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Clear the FLEXIO MCULCD multiple beats write mode configuration.
- void [FLEXIO_MCULCD_SetSingleBeatReadConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Configures the FLEXIO MCULCD to multiple beats read mode.
- void [FLEXIO_MCULCD_ClearSingleBeatReadConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Clear the FLEXIO MCULCD multiple beats read mode configuration.
- void [FLEXIO_MCULCD_SetMultiBeatsWriteConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Configures the FLEXIO MCULCD to multiple beats write mode.
- void [FLEXIO_MCULCD_ClearMultiBeatsWriteConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Clear the FLEXIO MCULCD multiple beats write mode configuration.
- void [FLEXIO_MCULCD_SetMultiBeatsReadConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Configures the FLEXIO MCULCD to multiple beats read mode.
- void [FLEXIO_MCULCD_ClearMultiBeatsReadConfig](#) ([FLEXIO_MCULCD_Type](#) *base)
Clear the FLEXIO MCULCD multiple beats read mode configuration.
- static void [FLEXIO_MCULCD_Enable](#) ([FLEXIO_MCULCD_Type](#) *base, bool enable)
Enables/disables the FlexIO MCULCD module operation.
- uint32_t [FLEXIO_MCULCD_ReadData](#) ([FLEXIO_MCULCD_Type](#) *base)
Read data from the FLEXIO MCULCD RX shifter buffer.
- static void [FLEXIO_MCULCD_WriteData](#) ([FLEXIO_MCULCD_Type](#) *base, uint32_t data)
Write data into the FLEXIO MCULCD TX shifter buffer.
- static void [FLEXIO_MCULCD_StartTransfer](#) ([FLEXIO_MCULCD_Type](#) *base)
Assert the nCS to start transfer.
- static void [FLEXIO_MCULCD_StopTransfer](#) ([FLEXIO_MCULCD_Type](#) *base)
De-assert the nCS to stop transfer.
- void [FLEXIO_MCULCD_WaitTransmitComplete](#) (void)
Wait for transmit data send out finished.

- void `FLEXIO_MCULCD_WriteCommandBlocking` (`FLEXIO_MCULCD_Type` *base, uint32_t command)
Send command in blocking way.
- void `FLEXIO_MCULCD_WriteDataArrayBlocking` (`FLEXIO_MCULCD_Type` *base, const void *data, size_t size)
Send data array in blocking way.
- void `FLEXIO_MCULCD_ReadDataArrayBlocking` (`FLEXIO_MCULCD_Type` *base, void *data, size_t size)
Read data into array in blocking way.
- void `FLEXIO_MCULCD_WriteSameValueBlocking` (`FLEXIO_MCULCD_Type` *base, uint32_t sameValue, size_t size)
Send the same value many times in blocking way.
- void `FLEXIO_MCULCD_TransferBlocking` (`FLEXIO_MCULCD_Type` *base, `flexio_mculcd_transfer_t` *xfer)
Performs a polling transfer.

Transactional

- `status_t FLEXIO_MCULCD_TransferCreateHandle` (`FLEXIO_MCULCD_Type` *base, `flexio_mculcd_handle_t` *handle, `flexio_mculcd_transfer_callback_t` callback, void *userData)
Initializes the FlexIO MCULCD handle, which is used in transactional functions.
- `status_t FLEXIO_MCULCD_TransferNonBlocking` (`FLEXIO_MCULCD_Type` *base, `flexio_mculcd_handle_t` *handle, `flexio_mculcd_transfer_t` *xfer)
Transfer data using IRQ.
- void `FLEXIO_MCULCD_TransferAbort` (`FLEXIO_MCULCD_Type` *base, `flexio_mculcd_handle_t` *handle)
Aborts the data transfer, which used IRQ.
- `status_t FLEXIO_MCULCD_TransferGetCount` (`FLEXIO_MCULCD_Type` *base, `flexio_mculcd_handle_t` *handle, size_t *count)
Gets the data transfer status which used IRQ.
- void `FLEXIO_MCULCD_TransferHandleIRQ` (void *base, void *handle)
FlexIO MCULCD IRQ handler function.

32.6.3 Data Structure Documentation

32.6.3.1 struct FLEXIO_MCULCD_Type

Data Fields

- `FLEXIO_Type` * `flexioBase`
FlexIO base pointer.
- `flexio_mculcd_bus_t` `busType`
The bus type, 8080 or 6800.
- uint8_t `dataPinStartIndex`
Start index of the data pin, the FlexIO pin dataPinStartIndex to (dataPinStartIndex + FLEXIO_MCULCD_DATA_BUS_WIDTH - 1) will be used for data transfer.

- uint8_t [ENWRPinIndex](#)
Pin select for WR(8080 mode), EN(6800 mode).
- uint8_t [RDPinIndex](#)
Pin select for RD(8080 mode), not used in 6800 mode.
- uint8_t [txShifterStartIndex](#)
Start index of shifters used for data write, it must be 0 or 4.
- uint8_t [txShifterEndIndex](#)
End index of shifters used for data write.
- uint8_t [rxShifterStartIndex](#)
Start index of shifters used for data read.
- uint8_t [rxShifterEndIndex](#)
End index of shifters used for data read, it must be 3 or 7.
- uint8_t [timerIndex](#)
Timer index used in FlexIO MCULCD.
- [flexio_mculcd_pin_func_t setCSPin](#)
Function to set or clear the CS pin.
- [flexio_mculcd_pin_func_t setRSPin](#)
Function to set or clear the RS pin.
- [flexio_mculcd_pin_func_t setRDWRPin](#)
Function to set or clear the RD/WR pin, only used in 6800 mode.

Field Documentation

(1) **FLEXIO_Type* FLEXIO_MCULCD_Type::flexioBase**

(2) **flexio_mculcd_bus_t FLEXIO_MCULCD_Type::busType**

(3) **uint8_t FLEXIO_MCULCD_Type::dataPinStartIndex**

Only support data bus width 8 and 16.

(4) **uint8_t FLEXIO_MCULCD_Type::ENWRPinIndex**

(5) **uint8_t FLEXIO_MCULCD_Type::RDPinIndex**

(6) **uint8_t FLEXIO_MCULCD_Type::txShifterStartIndex**

(7) **uint8_t FLEXIO_MCULCD_Type::txShifterEndIndex**

(8) **uint8_t FLEXIO_MCULCD_Type::rxShifterStartIndex**

(9) **uint8_t FLEXIO_MCULCD_Type::rxShifterEndIndex**

(10) **uint8_t FLEXIO_MCULCD_Type::timerIndex**

(11) **flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setCSPin**

(12) **flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRSPin**

(13) **flexio_mculcd_pin_func_t FLEXIO_MCULCD_Type::setRDWRPin**

32.6.3.2 struct flexio_mculcd_config_t

Data Fields

- bool [enable](#)
Enable/disable FlexIO MCULCD after configuration.
- bool [enableInDoze](#)
Enable/disable FlexIO operation in doze mode.
- bool [enableInDebug](#)
Enable/disable FlexIO operation in debug mode.
- bool [enableFastAccess](#)
Enable/disable fast access to FlexIO registers, fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.
- uint32_t [baudRate_Bps](#)
Baud rate in Bps.

Field Documentation

- (1) bool flexio_mculcd_config_t::enable
- (2) bool flexio_mculcd_config_t::enableInDoze
- (3) bool flexio_mculcd_config_t::enableInDebug
- (4) bool flexio_mculcd_config_t::enableFastAccess
- (5) uint32_t flexio_mculcd_config_t::baudRate_Bps

32.6.3.3 struct flexio_mculcd_transfer_t

Data Fields

- uint32_t [command](#)
Command to send.
- [flexio_mculcd_transfer_mode_t](#) mode
Transfer mode.
- uint32_t [dataAddrOrSameValue](#)
When sending the same value for many times, this is the value to send.
- size_t [dataSize](#)
How many bytes to transfer.

Field Documentation

- (1) uint32_t flexio_mculcd_transfer_t::command
- (2) flexio_mculcd_transfer_mode_t flexio_mculcd_transfer_t::mode
- (3) uint32_t flexio_mculcd_transfer_t::dataAddrOrSameValue

When writing or reading array, this is the address of the data array.

(4) `size_t flexio_mculcd_transfer_t::dataSize`

32.6.3.4 struct `_flexio_mculcd_handle`

typedef for `flexio_mculcd_handle_t` in advance.

Data Fields

- `uint32_t dataAddrOrSameValue`
When sending the same value for many times, this is the value to send.
- `size_t dataCount`
Total count to be transferred.
- `volatile size_t remainingCount`
Remaining count to transfer.
- `volatile uint32_t state`
FlexIO MCULCD internal state.
- `flexio_mculcd_transfer_callback_t completionCallback`
FlexIO MCULCD transfer completed callback.
- `void * userData`
Callback parameter.

Field Documentation

(1) `uint32_t flexio_mculcd_handle_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

(2) `size_t flexio_mculcd_handle_t::dataCount`

(3) `volatile size_t flexio_mculcd_handle_t::remainingCount`

(4) `volatile uint32_t flexio_mculcd_handle_t::state`

(5) `flexio_mculcd_transfer_callback_t flexio_mculcd_handle_t::completionCallback`

(6) `void* flexio_mculcd_handle_t::userData`

32.6.4 Macro Definition Documentation

32.6.4.1 `#define FSL_FLEXIO_MCULCD_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))`

32.6.4.2 `#define FLEXIO_MCULCD_WAIT_COMPLETE_TIME 512`

Currently there is no method to detect whether the data has been sent out from the shifter, so the driver use a software delay for this. When the data is written to shifter buffer, the driver call the delay function to wait for the data shift out. If this value is too small, then the last few bytes might be lost when writing data using interrupt method or DMA method.

32.6.5 Typedef Documentation

32.6.5.1 typedef void(* flexio_mculcd_pin_func_t)(bool set)

32.6.5.2 typedef void(* flexio_mculcd_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_handle_t *handle, status_t status, void *userData)

When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

32.6.6 Enumeration Type Documentation

32.6.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_MCULCD_Idle FlexIO LCD is idle.
kStatus_FLEXIO_MCULCD_Busy FlexIO LCD is busy.
kStatus_FLEXIO_MCULCD_Error FlexIO LCD error occurred.

32.6.6.2 enum flexio_mculcd_pixel_format_t

Enumerator

kFLEXIO_MCULCD_RGB565 RGB565, 16-bit.
kFLEXIO_MCULCD_BGR565 BGR565, 16-bit.
kFLEXIO_MCULCD_RGB888 RGB888, 24-bit.
kFLEXIO_MCULCD_BGR888 BGR888, 24-bit.

32.6.6.3 enum flexio_mculcd_bus_t

Enumerator

kFLEXIO_MCULCD_8080 Using Intel 8080 bus.
kFLEXIO_MCULCD_6800 Using Motorola 6800 bus.

32.6.6.4 enum _flexio_mculcd_interrupt_enable

Enumerator

kFLEXIO_MCULCD_TxEmptyInterruptEnable Transmit buffer empty interrupt enable.
kFLEXIO_MCULCD_RxFullInterruptEnable Receive buffer full interrupt enable.

32.6.6.5 enum _flexio_mculcd_status_flags

Enumerator

kFLEXIO_MCULCD_TxEmptyFlag Transmit buffer empty flag.
kFLEXIO_MCULCD_RxFullFlag Receive buffer full flag.

32.6.6.6 enum _flexio_mculcd_dma_enable

Enumerator

kFLEXIO_MCULCD_TxDmaEnable Tx DMA request source.
kFLEXIO_MCULCD_RxDmaEnable Rx DMA request source.

32.6.6.7 enum flexio_mculcd_transfer_mode_t

Enumerator

kFLEXIO_MCULCD_ReadArray Read data into an array.
kFLEXIO_MCULCD_WriteArray Write data from an array.
kFLEXIO_MCULCD_WriteSameValue Write the same value many times.

32.6.7 Function Documentation

32.6.7.1 status_t FLEXIO_MCULCD_Init (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_config_t * *config*, uint32_t *srcClock_Hz*)

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO_MCU-LCD_GetDefaultConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>config</i>	Pointer to the flexio_mculcd_config_t structure.
<i>srcClock_Hz</i>	FlexIO source clock in Hz.

Return values

<i>kStatus_Success</i>	Initialization success.
<i>kStatus_InvalidArgument</i>	Initialization failed because of invalid argument.

32.6.7.2 void FLEXIO_MCULCD_Deinit (FLEXIO_MCULCD_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

32.6.7.3 void FLEXIO_MCULCD_GetDefaultConfig (flexio_mculcd_config_t * *config*)

The default configuration value is:

```
* config->enable = true;
* config->enableInDoze = false;
* config->enableInDebug = true;
* config->enableFastAccess = true;
* config->baudRate_Bps = 96000000U;
*
```

Parameters

<i>config</i>	Pointer to the flexio_mculcd_config_t structure.
---------------	--

32.6.7.4 uint32_t FLEXIO_MCULCD_GetStatusFlags (FLEXIO_MCULCD_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

Returns

status flag; OR'ed value or the [_flexio_mculcd_status_flags](#).

Note

Don't use this function with DMA APIs.

32.6.7.5 void FLEXIO_MCULCD_ClearStatusFlags (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>mask</i>	Status to clear, it is the OR'ed value of _flexio_mculcd_status_flags .

Note

Don't use this function with DMA APIs.

32.6.7.6 void FLEXIO_MCULCD_EnableInterrupts (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

This function enables the FlexIO MCULCD interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>mask</i>	Interrupts to enable, it is the OR'ed value of _flexio_mculcd_interrupt_enable .

32.6.7.7 void FLEXIO_MCULCD_DisableInterrupts (FLEXIO_MCULCD_Type * *base*, uint32_t *mask*)

This function disables the FlexIO MCULCD interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>mask</i>	Interrupts to disable, it is the OR'ed value of _flexio_mculcd_interrupt_enable .

32.6.7.8 static void FLEXIO_MCULCD_EnableTxDMA (FLEXIO_MCULCD_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

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

32.6.7.9 static void FLEXIO_MCULCD_EnableRxDMA (FLEXIO_MCULCD_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>enable</i>	True means enable DMA, false means disable DMA.

32.6.7.10 static uint32_t FLEXIO_MCULCD_GetTxDataRegisterAddress (FLEXIO_MCULCD_Type * *base*) [inline], [static]

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

Returns

FlexIO MCULCD transmit data register address.

32.6.7.11 static uint32_t FLEXIO_MCULCD_GetRxDataRegisterAddress (FLEXIO_MCULCD_Type * *base*) [inline], [static]

This function returns the MCULCD data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

Returns

FlexIO MCULCD receive data register address.

32.6.7.12 status_t FLEXIO_MCULCD_SetBaudRate (FLEXIO_MCULCD_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>baudRate_Bps</i>	Desired baud rate.
<i>srcClock_Hz</i>	FLEXIO clock frequency in Hz.

Return values

<i>kStatus_Success</i>	Set successfully.
<i>kStatus_InvalidArgument</i>	Could not set the baud rate.

32.6.7.13 void FLEXIO_MCULCD_SetSingleBeatWriteConfig (FLEXIO_MCULCD_Type * *base*)

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by [FLEXIO_MCULCD_ClearSingleBeatWriteConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.14 void FLEXIO_MCULCD_ClearSingleBeatWriteConfig (FLEXIO_MCULCD_Type * *base*)

Clear the write configuration set by [FLEXIO_MCULCD_SetSingleBeatWriteConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.15 void FLEXIO_MCULCD_SetSingleBeatReadConfig (FLEXIO_MCULCD_Type * *base*)

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by [FLEXIO_MCULCD_ClearSingleBeatReadConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.16 void FLEXIO_MCULCD_ClearSingleBeatReadConfig (FLEXIO_MCULCD_Type * *base*)

Clear the read configuration set by [FLEXIO_MCULCD_SetSingleBeatReadConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.17 void FLEXIO_MCULCD_SetMultiBeatsWriteConfig (FLEXIO_MCULCD_Type * *base*)

At the begining multiple beats write operation, the FLEXIO MCULCD is configured to multiple beats write mode using this function. After write operation, the configuration is cleared by [FLEXIO_MCULCD_ClearMultBeatsWriteConfig](#).

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.18 void FLEXIO_MCULCD_ClearMultiBeatsWriteConfig (FLEXIO_MCULCD_Type * *base*)

Clear the write configuration set by FLEXIO_MCULCD_SetMultBeatsWriteConfig.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.19 void FLEXIO_MCULCD_SetMultiBeatsReadConfig (FLEXIO_MCULCD_Type * *base*)

At the begining or multiple beats read operation, the FLEXIO MCULCD is configured to multiple beats read mode using this function. After read operation, the configuration is cleared by FLEXIO_MCULCD_ClearMultBeatsReadConfig.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.20 void FLEXIO_MCULCD_ClearMultiBeatsReadConfig (FLEXIO_MCULCD_Type * *base*)

Clear the read configuration set by FLEXIO_MCULCD_SetMultBeatsReadConfig.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
-------------	---

Note

This is an internal used function, upper layer should not use.

32.6.7.21 static void FLEXIO_MCULCD_Enable (FLEXIO_MCULCD_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type .
<i>enable</i>	True to enable, false does not have any effect.

32.6.7.22 uint32_t FLEXIO_MCULCD_ReadData (FLEXIO_MCULCD_Type * *base*)

Read data from the RX shift buffer directly, it does no check whether the buffer is empty or not.

If the data bus width is 8-bit:

```
* uint8_t value;
* value = (uint8_t)FLEXIO_MCULCD_ReadData(base);
*
```

If the data bus width is 16-bit:

```
* uint16_t value;
* value = (uint16_t)FLEXIO_MCULCD_ReadData(base);
*
```

Note

This function returns the RX shifter buffer value (32-bit) directly. The return value should be converted according to data bus width.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

Returns

The data read out.

Note

Don't use this function with DMA APIs.

32.6.7.23 static void FLEXIO_MCULCD_WriteData (FLEXIO_MCULCD_Type * *base*, uint32_t *data*) [inline], [static]

Write data into the TX shift buffer directly, it does no check whether the buffer is full or not.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>data</i>	The data to write.

Note

Don't use this function with DMA APIs.

32.6.7.24 static void FLEXIO_MCULCD_StartTransfer (FLEXIO_MCULCD_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

32.6.7.25 static void FLEXIO_MCULCD_StopTransfer (FLEXIO_MCULCD_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
-------------	--

32.6.7.26 void FLEXIO_MCULCD_WaitTransmitComplete (void)

Currently there is no effective method to wait for the data send out from the shiter, so here use a while loop to wait.

Note

This is an internal used function.

32.6.7.27 void FLEXIO_MCULCD_WriteCommandBlocking (FLEXIO_MCULCD_Type * *base*, uint32_t *command*)

This function sends the command and returns when the command has been sent out.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>command</i>	The command to send.

32.6.7.28 void FLEXIO_MCULCD_WriteDataArrayBlocking (FLEXIO_MCULCD_Type * *base*, const void * *data*, size_t *size*)

This function sends the data array and returns when the data sent out.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>data</i>	The data array to send.
<i>size</i>	How many bytes to write.

32.6.7.29 void FLEXIO_MCULCD_ReadDataArrayBlocking (FLEXIO_MCULCD_Type * *base*, void * *data*, size_t *size*)

This function reads the data into array and returns when the data read finished.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>data</i>	The array to save the data.
<i>size</i>	How many bytes to read.

32.6.7.30 void FLEXIO_MCULCD_WriteSameValueBlocking (FLEXIO_MCULCD_Type * *base*, uint32_t *sameValue*, size_t *size*)

This function sends the same value many times. It could be used to clear the LCD screen. If the data bus width is 8, this function will send LSB 8 bits of *sameValue* for *size* times. If the data bus is 16, this function will send LSB 16 bits of *sameValue* for *size* / 2 times.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>sameValue</i>	The same value to send.
<i>size</i>	How many bytes to send.

32.6.7.31 void FLEXIO_MCULCD_TransferBlocking (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_transfer_t * *xfer*)

Note

The API does not return until the transfer finished.

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type structure.
<i>xfer</i>	pointer to flexio_mculcd_transfer_t structure.

32.6.7.32 status_t FLEXIO_MCULCD_TransferCreateHandle (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, flexio_mculcd_transfer_callback_t *callback*, void * *userData*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

32.6.7.33 status_t FLEXIO_MCULCD_TransferNonBlocking (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer*)

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
<i>xfer</i>	FlexIO MCULCD transfer structure. See flexio_mculcd_transfer_t .

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_FLEXIO_MCULCD_Busy</i>	MCULCD is busy with another transfer.

32.6.7.34 void FLEXIO_MCULCD_TransferAbort (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

32.6.7.35 status_t FLEXIO_MCULCD_TransferGetCount (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.
<i>count</i>	How many bytes transferred so far by the non-blocking transaction.

Return values

<i>kStatus_Success</i>	Get the transferred count Successfully.
<i>kStatus_NoTransferInProgress</i>	No transfer in process.

32.6.7.36 void FLEXIO_MCULCD_TransferHandleIRQ (void * *base*, void * *handle*)

Parameters

<i>base</i>	Pointer to the FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to the flexio_mculcd_handle_t structure to store the transfer state.

32.6.8 FlexIO eDMA MCU Interface LCD Driver

SDK provide eDMA transactional APIs to transfer data using eDMA, the eDMA method is similar with interrupt transactional method.

32.6.8.1 Overview

Note

eDMA transactional functions use multiple beats method for better performance, in contrast, the blocking functions and interrupt functions use single beat method. The function [FLEXIO_MCULCD_ReadData](#), [FLEXIO_MCULCD_WriteData](#), [FLEXIO_MCULCD_GetStatusFlags](#), and [FLEXIO_MCULCD_ClearStatusFlags](#) are only used for single beat case, so don't use these functions to work together with eDMA functions.

FlexIO eDMA MCU Interface LCD Driver

FLEXIO LCD send/receive using a DMA method

```
flexio_MCULCD_edma_handle_t handle;
volatile bool completeFlag = false;
edma_handle_t rxEdmaHandle;
edma_handle_t txEdmaHandle;

void flexioLcdCallback(FLEXIO_MCULCD_Type *base, flexio_MCULCD_edma_handle_t *handle,
    status_t status, void *userData)
{
    if (kStatus_FLEXIO_MCULCD_Idle == status)
    {
        completeFlag = true;
    }
}

void main(void)
{
    // Create the edma Handle.
    EDMA_CreateHandle(&rxEdmaHandle, DMA0, channel);
    EDMA_CreateHandle(&txEdmaHandle, DMA0, channel);

    // Configure the DMAMUX.
    // ...
    // rxEdmaHandle should use the last FlexIO RX shifters as DMA request source.
    // txEdmaHandle should use the first FlexIO TX shifters as DMA request source.

    // Init the FlexIO LCD driver.
    FLEXIO_MCULCD_Init(...);

    // Create the transactional handle.
    FLEXIO_MCULCD_TransferCreateHandleEDMA(&flexioLcdDev, &handle,
        flexioLcdCallback, NULL, &txEdmaHandle, &rxEdmaHandle);

    xfer.command = command2;
    xfer.mode = kFLEXIO_MCULCD_WriteArray;
    xfer.dataAddrOrSameValue = (uint32_t)dataToSend;
    xfer.dataCount = sizeof(dataToSend);
    completeFlag = false;
    FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);
}
```

```

while (!completeFlag)
{
}

xfer.command = command2;
xfer.mode = kFLEXIO_MCULCD_WriteSameValue;
xfer.dataAddrOrSameValue = value;
xfer.dataCount = 1000;
completeFlag = false;
FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

while (!completeFlag)
{
}

xfer.command = command3;
xfer.mode = kFLEXIO_MCULCD_ReadArray;
xfer.dataAddrOrSameValue = (uint32_t)dataToReceive;
xfer.dataCount = sizeof(dataToReceive);
completeFlag = false;
FLEXIO_MCULCD_TransferEDMA(&flexioLcdDev, &handle, &xfer);

while (!completeFlag)
{
}
}

```

Data Structures

- struct [flexio_mculcd_edma_handle_t](#)
FlexIO MCULCD eDMA transfer handle, users should not touch the content of the handle. [More...](#)

Macros

- #define [FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 4))
FlexIO MCULCD EDMA driver version.

Typedefs

- typedef void(* [flexio_mculcd_edma_transfer_callback_t](#))([FLEXIO_MCULCD_Type](#) *base, [flexio_mculcd_edma_handle_t](#) *handle, [status_t](#) status, void *userData)
FlexIO MCULCD master callback for transfer complete.

eDMA Transactional

- [status_t](#) [FLEXIO_MCULCD_TransferCreateHandleEDMA](#) ([FLEXIO_MCULCD_Type](#) *base, [flexio_mculcd_edma_handle_t](#) *handle, [flexio_mculcd_edma_transfer_callback_t](#) callback, void *userData, [edma_handle_t](#) *txDmaHandle, [edma_handle_t](#) *rxDmaHandle)
Initializes the FLEXIO MCULCD master eDMA handle.
- [status_t](#) [FLEXIO_MCULCD_TransferEDMA](#) ([FLEXIO_MCULCD_Type](#) *base, [flexio_mculcd_edma_handle_t](#) *handle, [flexio_mculcd_transfer_t](#) *xfer)
Performs a non-blocking FlexIO MCULCD transfer using eDMA.

- void [FLEXIO_MCULCD_TransferAbortEDMA](#) ([FLEXIO_MCULCD_Type](#) *base, [flexio_mculcd_edma_handle_t](#) *handle)
Aborts a FlexIO MCULCD transfer using eDMA.
- [status_t FLEXIO_MCULCD_TransferGetCountEDMA](#) ([FLEXIO_MCULCD_Type](#) *base, [flexio_mculcd_edma_handle_t](#) *handle, [size_t](#) *count)
Gets the remaining bytes for FlexIO MCULCD eDMA transfer.

32.6.8.2 Data Structure Documentation

32.6.8.2.1 struct flexio_mculcd_edma_handle

typedef for [flexio_mculcd_edma_handle_t](#) in advance.

Data Fields

- [FLEXIO_MCULCD_Type](#) * base
Pointer to the [FLEXIO_MCULCD_Type](#).
- [uint8_t txShifterNum](#)
Number of shifters used for TX.
- [uint8_t rxShifterNum](#)
Number of shifters used for RX.
- [uint32_t minorLoopBytes](#)
eDMA transfer minor loop bytes.
- [edma_modulo_t txEdmaModulo](#)
Modulo value for the FlexIO shifter buffer access.
- [edma_modulo_t rxEdmaModulo](#)
Modulo value for the FlexIO shifter buffer access.
- [uint32_t dataAddrOrSameValue](#)
When sending the same value for many times, this is the value to send.
- [size_t dataCount](#)
Total count to be transferred.
- [volatile size_t remainingCount](#)
Remaining count still not transferred.
- [volatile uint32_t state](#)
FlexIO MCULCD driver internal state.
- [edma_handle_t](#) * [txDmaHandle](#)
DMA handle for MCULCD TX.
- [edma_handle_t](#) * [rxDmaHandle](#)
DMA handle for MCULCD RX.
- [flexio_mculcd_edma_transfer_callback_t completionCallback](#)
Callback for MCULCD DMA transfer.
- void * [userData](#)
User Data for MCULCD DMA callback.

Field Documentation

(1) [FLEXIO_MCULCD_Type](#)* [flexio_mculcd_edma_handle_t::base](#)

- (2) `uint8_t flexio_mculcd_edma_handle_t::txShifterNum`
- (3) `uint8_t flexio_mculcd_edma_handle_t::rxShifterNum`
- (4) `uint32_t flexio_mculcd_edma_handle_t::minorLoopBytes`
- (5) `edma_modulo_t flexio_mculcd_edma_handle_t::txEdmaModulo`
- (6) `edma_modulo_t flexio_mculcd_edma_handle_t::rxEdmaModulo`
- (7) `uint32_t flexio_mculcd_edma_handle_t::dataAddrOrSameValue`

When writing or reading array, this is the address of the data array.

- (8) `size_t flexio_mculcd_edma_handle_t::dataCount`
- (9) `volatile size_t flexio_mculcd_edma_handle_t::remainingCount`
- (10) `volatile uint32_t flexio_mculcd_edma_handle_t::state`

32.6.8.3 Macro Definition Documentation

32.6.8.3.1 `#define FSL_FLEXIO_MCULCD_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

32.6.8.4 Typedef Documentation

32.6.8.4.1 `typedef void(* flexio_mculcd_edma_transfer_callback_t)(FLEXIO_MCULCD_Type *base, flexio_mculcd_edma_handle_t *handle, status_t status, void *userData)`

When transfer finished, the callback function is called and returns the `status` as `kStatus_FLEXIO_MCULCD_Idle`.

32.6.8.5 Function Documentation

32.6.8.5.1 `status_t FLEXIO_MCULCD_TransferCreateHandleEDMA (FLEXIO_MCULCD_Type * base, flexio_mculcd_edma_handle_t * handle, flexio_mculcd_edma_transfer_callback_t callback, void * userData, edma_handle_t * txDmaHandle, edma_handle_t * rxDmaHandle)`

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

Parameters

<i>base</i>	Pointer to FLEXIO_MCULCD_Type structure.
<i>handle</i>	Pointer to flexio_mculcd_edma_handle_t structure to store the transfer state.
<i>callback</i>	MCULCD transfer complete callback, NULL means no callback.
<i>userData</i>	callback function parameter.
<i>txDmaHandle</i>	User requested eDMA handle for FlexIO MCULCD eDMA TX, the DMA request source of this handle should be the first of TX shifters.
<i>rxDmaHandle</i>	User requested eDMA handle for FlexIO MCULCD eDMA RX, the DMA request source of this handle should be the last of RX shifters.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
------------------------	---------------------------------

32.6.8.5.2 status_t FLEXIO_MCULCD_TransferEDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_edma_handle_t * *handle*, flexio_mculcd_transfer_t * *xfer*)

This function returns immediately after transfer initiates. To check whether the transfer is completed, user could:

1. Use the transfer completed callback;
2. Polling function FLEXIO_MCULCD_GetTransferCountEDMA

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type structure.
<i>handle</i>	pointer to flexio_mculcd_edma_handle_t structure to store the transfer state.
<i>xfer</i>	Pointer to FlexIO MCULCD transfer structure.

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_FLEXIO_MCULCD_Busy</i>	FlexIO MCULCD is not idle, it is running another transfer.

32.6.8.5.3 void FLEXIO_MCULCD_TransferAbortEDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_edma_handle_t * *handle*)

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type structure.
<i>handle</i>	FlexIO MCULCD eDMA handle pointer.

32.6.8.5.4 status_t FLEXIO_MCULCD_TransferGetCountEDMA (FLEXIO_MCULCD_Type * *base*, flexio_mculcd_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	pointer to FLEXIO_MCULCD_Type structure.
<i>handle</i>	FlexIO MCULCD eDMA handle pointer.
<i>count</i>	Number of count transferred so far by the eDMA transaction.

Return values

<i>kStatus_Success</i>	Get the transferred count Successfully.
<i>kStatus_NoTransferInProgress</i>	No transfer in process.

32.7 FlexIO SPI Driver

32.7.1 Overview

The MCUXpresso SDK provides a peripheral driver for an SPI function using the Flexible I/O module of MCUXpresso SDK devices.

FlexIO SPI driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for FlexIO SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the FlexIO SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the [FLEXIO_SPI_Type](#) *base as the first parameter. FlexIO SPI functional operation groups provide the functional API set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs can satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_spi_master_handle_t/flexio_spi_slave_handle_t` as the second parameter. Initialize the handle by calling the [FLEXIO_SPI_MasterTransferCreateHandle\(\)](#) or [FLEXIO_SPI_SlaveTransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [FLEXIO_SPI_MasterTransferNonBlocking\(\)](#)/[FLEXIO_SPI_SlaveTransferNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the `kStatus_FLEXIO_SPI_Idle` status.

Note that the FlexIO SPI slave driver only supports discontinuous PCS access, which is a limitation. The FlexIO SPI slave driver can support continuous PCS, but the slave cannot adapt discontinuous and continuous PCS automatically. Users can change the timer disable mode in [FLEXIO_SPI_SlaveInit](#) manually, from `kFLEXIO_TimerDisableOnTimerCompare` to `kFLEXIO_TimerDisableNever` to enable a discontinuous PCS access. Only `CPHA = 0` is supported.

32.7.2 Typical use case

32.7.2.1 FlexIO SPI send/receive using an interrupt method

```
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];

void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_handle_t *handle
    , status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
}
```

```

}

void main(void)
{
    //...
    flexio_spi_transfer_t xfer = {0};
    flexio_spi_master_config_t userConfig;

    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);
    userConfig.baudRate_Bps = 500000U;

    spiDev.flexioBase = BOARD_FLEXIO_BASE;
    spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
    spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
    spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
    spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
    spiDev.shifterIndex[0] = 0U;
    spiDev.shifterIndex[1] = 1U;
    spiDev.timerIndex[0] = 0U;
    spiDev.timerIndex[1] = 1U;

    FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

    xfer.txData = srcBuff;
    xfer.rxData = destBuff;
    xfer.dataSize = BUFFER_SIZE;
    xfer.flags = kFLEXIO_SPI_8bitMsb;
    FLEXIO_SPI_MasterTransferCreateHandle(&spiDev, &g_spiHandle,
        FLEXIO_SPI_MasterUserCallback, NULL);
    FLEXIO_SPI_MasterTransferNonBlocking(&spiDev, &g_spiHandle, &xfer);

    // Send finished.
    while (!txFinished)
    {
        // ...
    }
}

```

32.7.2.2 FlexIO_SPI Send/Receive in DMA way

```

dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
flexio_spi_master_handle_t g_spiHandle;
FLEXIO_SPI_Type spiDev;
volatile bool txFinished;
static uint8_t srcBuff[BUFFER_SIZE];
static uint8_t destBuff[BUFFER_SIZE];
void FLEXIO_SPI_MasterUserCallback(FLEXIO_SPI_Type *base, flexio_spi_master_dma_handle_t
    *handle, status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_SPI_Idle == status)
    {
        txFinished = true;
    }
}

void main(void)
{
    flexio_spi_transfer_t xfer = {0};
    flexio_spi_master_config_t userConfig;

    FLEXIO_SPI_MasterGetDefaultConfig(&userConfig);

```

```

userConfig.baudRate_Bps = 500000U;

spiDev.flexioBase = BOARD_FLEXIO_BASE;
spiDev.SDOPinIndex = FLEXIO_SPI_MOSI_PIN;
spiDev.SDIPinIndex = FLEXIO_SPI_MISO_PIN;
spiDev.SCKPinIndex = FLEXIO_SPI_SCK_PIN;
spiDev.CSnPinIndex = FLEXIO_SPI_CSn_PIN;
spiDev.shifterIndex[0] = 0U;
spiDev.shifterIndex[1] = 1U;
spiDev.timerIndex[0] = 0U;
spiDev.timerIndex[1] = 1U;

/* Init DMAMUX. */
DMAMUX_Init(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR)

/* Init the DMA/EDMA module */
#if defined(FSL_FEATURE_SOC_DMA_COUNT) && FSL_FEATURE_SOC_DMA_COUNT > 0U
    DMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR);
    DMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL);
    DMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#if defined(FSL_FEATURE_SOC_EDMA_COUNT) && FSL_FEATURE_SOC_EDMA_COUNT > 0U
    edma_config_t edmaConfig;

    EDMA_GetDefaultConfig(&edmaConfig);
    EDMA_Init(EXAMPLE_FLEXIO_SPI_DMA_BASEADDR, &edmaConfig);
    EDMA_CreateHandle(&txHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
FLEXIO_SPI_TX_DMA_CHANNEL);
    EDMA_CreateHandle(&rxHandle, EXAMPLE_FLEXIO_SPI_DMA_BASEADDR,
FLEXIO_SPI_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

    dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
shifterIndex[0]);
    dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + spiDev.
shifterIndex[1]);

    /* Requests DMA channels for transmit and receive. */
    DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_TX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_tx);
    DMAMUX_SetSource(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR, FLEXIO_SPI_RX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_rx);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
FLEXIO_SPI_TX_DMA_CHANNEL);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_SPI_DMAMUX_BASEADDR,
FLEXIO_SPI_RX_DMA_CHANNEL);

    FLEXIO_SPI_MasterInit(&spiDev, &userConfig, FLEXIO_CLOCK_FREQUENCY);

/* Initializes the buffer. */
for (i = 0; i < BUFFER_SIZE; i++)
{
    srcBuff[i] = i;
}

/* Sends to the slave. */
xfer.txData = srcBuff;
xfer.rxData = destBuff;
xfer.dataSize = BUFFER_SIZE;
xfer.flags = kFLEXIO_SPI_8bitMsb;
FLEXIO_SPI_MasterTransferCreateHandleDMA(&spiDev, &g_spiHandle, FLEXIO_SPI_MasterUserCallback, NULL
, &g_spiTxDmaHandle, &g_spiRxDmaHandle);
FLEXIO_SPI_MasterTransferDMA(&spiDev, &g_spiHandle, &xfer);

// Send finished.
while (!txFinished)
{

```

```

    }
    // ...
}

```

Modules

- [FlexIO eDMA SPI Driver](#)

Data Structures

- struct [FLEXIO_SPI_Type](#)
Define FlexIO SPI access structure typedef. [More...](#)
- struct [flexio_spi_master_config_t](#)
Define FlexIO SPI master configuration structure. [More...](#)
- struct [flexio_spi_slave_config_t](#)
Define FlexIO SPI slave configuration structure. [More...](#)
- struct [flexio_spi_transfer_t](#)
Define FlexIO SPI transfer structure. [More...](#)
- struct [flexio_spi_master_handle_t](#)
Define FlexIO SPI handle structure. [More...](#)

Macros

- #define [FLEXIO_SPI_DUMMYDATA](#) (0xFFFFU)
FlexIO SPI dummy transfer data, the data is sent while txData is NULL.
- #define [SPI_RETRY_TIMES](#) 0U /* Define to zero means keep waiting until the flag is assert/deassert. */
Retry times for waiting flag.

Typedefs

- typedef [flexio_spi_master_handle_t](#) [flexio_spi_slave_handle_t](#)
Slave handle is the same with master handle.
- typedef void(* [flexio_spi_master_transfer_callback_t](#))(FLEXIO_SPI_Type *base, [flexio_spi_master_handle_t](#) *handle, [status_t](#) status, void *userData)
FlexIO SPI master callback for finished transmit.
- typedef void(* [flexio_spi_slave_transfer_callback_t](#))(FLEXIO_SPI_Type *base, [flexio_spi_slave_handle_t](#) *handle, [status_t](#) status, void *userData)
FlexIO SPI slave callback for finished transmit.

Enumerations

- enum {
 kStatus_FLEXIO_SPI_Busy = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 1),
 kStatus_FLEXIO_SPI_Idle = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 2),
 kStatus_FLEXIO_SPI_Error = MAKE_STATUS(kStatusGroup_FLEXIO_SPI, 3),
 kStatus_FLEXIO_SPI_Timeout }
 Error codes for the FlexIO SPI driver.
- enum flexio_spi_clock_phase_t {
 kFLEXIO_SPI_ClockPhaseFirstEdge = 0x0U,
 kFLEXIO_SPI_ClockPhaseSecondEdge = 0x1U }
 FlexIO SPI clock phase configuration.
- enum flexio_spi_shift_direction_t {
 kFLEXIO_SPI_MsbFirst = 0,
 kFLEXIO_SPI_LsbFirst = 1 }
 FlexIO SPI data shifter direction options.
- enum flexio_spi_data_bitcount_mode_t {
 kFLEXIO_SPI_8BitMode = 0x08U,
 kFLEXIO_SPI_16BitMode = 0x10U }
 FlexIO SPI data length mode options.
- enum _flexio_spi_interrupt_enable {
 kFLEXIO_SPI_TxEmptyInterruptEnable = 0x1U,
 kFLEXIO_SPI_RxFullInterruptEnable = 0x2U }
 Define FlexIO SPI interrupt mask.
- enum _flexio_spi_status_flags {
 kFLEXIO_SPI_TxBufferEmptyFlag = 0x1U,
 kFLEXIO_SPI_RxBufferFullFlag = 0x2U }
 Define FlexIO SPI status mask.
- enum _flexio_spi_dma_enable {
 kFLEXIO_SPI_TxDmaEnable = 0x1U,
 kFLEXIO_SPI_RxDmaEnable = 0x2U,
 kFLEXIO_SPI_DmaAllEnable = 0x3U }
 Define FlexIO SPI DMA mask.
- enum _flexio_spi_transfer_flags {
 kFLEXIO_SPI_8bitMsb = 0x1U,
 kFLEXIO_SPI_8bitLsb = 0x2U,
 kFLEXIO_SPI_16bitMsb = 0x9U,
 kFLEXIO_SPI_16bitLsb = 0xaU }
 Define FlexIO SPI transfer flags.

Driver version

- #define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
 FlexIO SPI driver version 2.2.1.

FlexIO SPI Configuration

- void **FLEXIO_SPI_MasterInit** (**FLEXIO_SPI_Type** *base, **flexio_spi_master_config_t** *masterConfig, uint32_t srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI master hardware, and configures the FlexIO SPI with FlexIO SPI master configuration.
- void **FLEXIO_SPI_MasterDeinit** (**FLEXIO_SPI_Type** *base)
Resets the FlexIO SPI timer and shifter config.
- void **FLEXIO_SPI_MasterGetDefaultConfig** (**flexio_spi_master_config_t** *masterConfig)
Gets the default configuration to configure the FlexIO SPI master.
- void **FLEXIO_SPI_SlaveInit** (**FLEXIO_SPI_Type** *base, **flexio_spi_slave_config_t** *slaveConfig)
Ungates the FlexIO clock, resets the FlexIO module, configures the FlexIO SPI slave hardware configuration, and configures the FlexIO SPI with FlexIO SPI slave configuration.
- void **FLEXIO_SPI_SlaveDeinit** (**FLEXIO_SPI_Type** *base)
Gates the FlexIO clock.
- void **FLEXIO_SPI_SlaveGetDefaultConfig** (**flexio_spi_slave_config_t** *slaveConfig)
Gets the default configuration to configure the FlexIO SPI slave.

Status

- uint32_t **FLEXIO_SPI_GetStatusFlags** (**FLEXIO_SPI_Type** *base)
Gets FlexIO SPI status flags.
- void **FLEXIO_SPI_ClearStatusFlags** (**FLEXIO_SPI_Type** *base, uint32_t mask)
Clears FlexIO SPI status flags.

Interrupts

- void **FLEXIO_SPI_EnableInterrupts** (**FLEXIO_SPI_Type** *base, uint32_t mask)
Enables the FlexIO SPI interrupt.
- void **FLEXIO_SPI_DisableInterrupts** (**FLEXIO_SPI_Type** *base, uint32_t mask)
Disables the FlexIO SPI interrupt.

DMA Control

- void **FLEXIO_SPI_EnableDMA** (**FLEXIO_SPI_Type** *base, uint32_t mask, bool enable)
Enables/disables the FlexIO SPI transmit DMA.
- static uint32_t **FLEXIO_SPI_GetTxDataRegisterAddress** (**FLEXIO_SPI_Type** *base, **flexio_spi_shift_direction_t** direction)
Gets the FlexIO SPI transmit data register address for MSB first transfer.
- static uint32_t **FLEXIO_SPI_GetRxDataRegisterAddress** (**FLEXIO_SPI_Type** *base, **flexio_spi_shift_direction_t** direction)
Gets the FlexIO SPI receive data register address for the MSB first transfer.

Bus Operations

- static void `FLEXIO_SPI_Enable` (`FLEXIO_SPI_Type` *base, bool enable)
Enables/disables the FlexIO SPI module operation.
- void `FLEXIO_SPI_MasterSetBaudRate` (`FLEXIO_SPI_Type` *base, uint32_t baudRate_Bps, uint32_t srcClockHz)
Sets baud rate for the FlexIO SPI transfer, which is only used for the master.
- static void `FLEXIO_SPI_WriteData` (`FLEXIO_SPI_Type` *base, flexio_spi_shift_direction_t direction, uint16_t data)
Writes one byte of data, which is sent using the MSB method.
- static uint16_t `FLEXIO_SPI_ReadData` (`FLEXIO_SPI_Type` *base, flexio_spi_shift_direction_t direction)
Reads 8 bit/16 bit data.
- status_t `FLEXIO_SPI_WriteBlocking` (`FLEXIO_SPI_Type` *base, flexio_spi_shift_direction_t direction, const uint8_t *buffer, size_t size)
Sends a buffer of data bytes.
- status_t `FLEXIO_SPI_ReadBlocking` (`FLEXIO_SPI_Type` *base, flexio_spi_shift_direction_t direction, uint8_t *buffer, size_t size)
Receives a buffer of bytes.
- status_t `FLEXIO_SPI_MasterTransferBlocking` (`FLEXIO_SPI_Type` *base, flexio_spi_transfer_t *xfer)
Receives a buffer of bytes.

Transactional

- status_t `FLEXIO_SPI_MasterTransferCreateHandle` (`FLEXIO_SPI_Type` *base, flexio_spi_master_handle_t *handle, flexio_spi_master_transfer_callback_t callback, void *userData)
Initializes the FlexIO SPI Master handle, which is used in transactional functions.
- status_t `FLEXIO_SPI_MasterTransferNonBlocking` (`FLEXIO_SPI_Type` *base, flexio_spi_master_handle_t *handle, flexio_spi_transfer_t *xfer)
Master transfer data using IRQ.
- void `FLEXIO_SPI_MasterTransferAbort` (`FLEXIO_SPI_Type` *base, flexio_spi_master_handle_t *handle)
Aborts the master data transfer, which used IRQ.
- status_t `FLEXIO_SPI_MasterTransferGetCount` (`FLEXIO_SPI_Type` *base, flexio_spi_master_handle_t *handle, size_t *count)
Gets the data transfer status which used IRQ.
- void `FLEXIO_SPI_MasterTransferHandleIRQ` (void *spiType, void *spiHandle)
FlexIO SPI master IRQ handler function.
- status_t `FLEXIO_SPI_SlaveTransferCreateHandle` (`FLEXIO_SPI_Type` *base, flexio_spi_slave_handle_t *handle, flexio_spi_slave_transfer_callback_t callback, void *userData)
Initializes the FlexIO SPI Slave handle, which is used in transactional functions.
- status_t `FLEXIO_SPI_SlaveTransferNonBlocking` (`FLEXIO_SPI_Type` *base, flexio_spi_slave_handle_t *handle, flexio_spi_transfer_t *xfer)
Slave transfer data using IRQ.
- static void `FLEXIO_SPI_SlaveTransferAbort` (`FLEXIO_SPI_Type` *base, flexio_spi_slave_handle_t *handle)
Aborts the slave data transfer which used IRQ, share same API with master.

- static `status_t FLEXIO_SPI_SlaveTransferGetCount` (`FLEXIO_SPI_Type *base`, `flexio_spi_slave_handle_t *handle`, `size_t *count`)
Gets the data transfer status which used IRQ, share same API with master.
- void `FLEXIO_SPI_SlaveTransferHandleIRQ` (`void *spiType`, `void *spiHandle`)
FlexIO SPI slave IRQ handler function.

32.7.3 Data Structure Documentation

32.7.3.1 struct FLEXIO_SPI_Type

Data Fields

- `FLEXIO_Type * flexioBase`
FlexIO base pointer.
- `uint8_t SDOPinIndex`
Pin select for data output.
- `uint8_t SDIPinIndex`
Pin select for data input.
- `uint8_t SCKPinIndex`
Pin select for clock.
- `uint8_t CSnPinIndex`
Pin select for enable.
- `uint8_t shifterIndex [2]`
Shifter index used in FlexIO SPI.
- `uint8_t timerIndex [2]`
Timer index used in FlexIO SPI.

Field Documentation

- (1) `FLEXIO_Type* FLEXIO_SPI_Type::flexioBase`
- (2) `uint8_t FLEXIO_SPI_Type::SDOPinIndex`
- (3) `uint8_t FLEXIO_SPI_Type::SDIPinIndex`
- (4) `uint8_t FLEXIO_SPI_Type::SCKPinIndex`
- (5) `uint8_t FLEXIO_SPI_Type::CSnPinIndex`
- (6) `uint8_t FLEXIO_SPI_Type::shifterIndex[2]`
- (7) `uint8_t FLEXIO_SPI_Type::timerIndex[2]`

32.7.3.2 struct flexio_spi_master_config_t

Data Fields

- bool `enableMaster`
Enable/disable FlexIO SPI master after configuration.

- bool [enableInDoze](#)
Enable/disable FlexIO operation in doze mode.
- bool [enableInDebug](#)
Enable/disable FlexIO operation in debug mode.
- bool [enableFastAccess](#)
*Enable/disable fast access to FlexIO registers,
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32_t [baudRate_Bps](#)
Baud rate in Bps.
- [flexio_spi_clock_phase_t](#) phase
Clock phase.
- [flexio_spi_data_bitcount_mode_t](#) dataMode
8bit or 16bit mode.

Field Documentation

- (1) bool flexio_spi_master_config_t::enableMaster
- (2) bool flexio_spi_master_config_t::enableInDoze
- (3) bool flexio_spi_master_config_t::enableInDebug
- (4) bool flexio_spi_master_config_t::enableFastAccess
- (5) uint32_t flexio_spi_master_config_t::baudRate_Bps
- (6) flexio_spi_clock_phase_t flexio_spi_master_config_t::phase
- (7) flexio_spi_data_bitcount_mode_t flexio_spi_master_config_t::dataMode

32.7.3.3 struct flexio_spi_slave_config_t

Data Fields

- bool [enableSlave](#)
Enable/disable FlexIO SPI slave after configuration.
- bool [enableInDoze](#)
Enable/disable FlexIO operation in doze mode.
- bool [enableInDebug](#)
Enable/disable FlexIO operation in debug mode.
- bool [enableFastAccess](#)
*Enable/disable fast access to FlexIO registers,
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- [flexio_spi_clock_phase_t](#) phase
Clock phase.
- [flexio_spi_data_bitcount_mode_t](#) dataMode
8bit or 16bit mode.

Field Documentation

- (1) bool flexio_spi_slave_config_t::enableSlave

- (2) `bool flexio_spi_slave_config_t::enableInDoze`
- (3) `bool flexio_spi_slave_config_t::enableInDebug`
- (4) `bool flexio_spi_slave_config_t::enableFastAccess`
- (5) `flexio_spi_clock_phase_t flexio_spi_slave_config_t::phase`
- (6) `flexio_spi_data_bitcount_mode_t flexio_spi_slave_config_t::dataMode`

32.7.3.4 struct flexio_spi_transfer_t

Data Fields

- `uint8_t * txData`
Send buffer.
- `uint8_t * rxData`
Receive buffer.
- `size_t dataSize`
Transfer bytes.
- `uint8_t flags`
FlexIO SPI control flag, MSB first or LSB first.

Field Documentation

- (1) `uint8_t* flexio_spi_transfer_t::txData`
- (2) `uint8_t* flexio_spi_transfer_t::rxData`
- (3) `size_t flexio_spi_transfer_t::dataSize`
- (4) `uint8_t flexio_spi_transfer_t::flags`

32.7.3.5 struct _flexio_spi_master_handle

typedef for `flexio_spi_master_handle_t` in advance.

Data Fields

- `uint8_t * txData`
Transfer buffer.
- `uint8_t * rxData`
Receive buffer.
- `size_t transferSize`
Total bytes to be transferred.
- volatile `size_t txRemainingBytes`
Send data remaining in bytes.
- volatile `size_t rxRemainingBytes`
Receive data remaining in bytes.
- volatile `uint32_t state`

- *FlexIO SPI internal state.*
uint8_t `bytePerFrame`
SPI mode, 2bytes or 1byte in a frame.
- `flexio_spi_shift_direction_t` `direction`
Shift direction.
- `flexio_spi_master_transfer_callback_t` `callback`
FlexIO SPI callback.
- void * `userData`
Callback parameter.

Field Documentation

- (1) uint8_t* `flexio_spi_master_handle_t::txData`
- (2) uint8_t* `flexio_spi_master_handle_t::rxData`
- (3) size_t `flexio_spi_master_handle_t::transferSize`
- (4) volatile size_t `flexio_spi_master_handle_t::txRemainingBytes`
- (5) volatile size_t `flexio_spi_master_handle_t::rxRemainingBytes`
- (6) volatile uint32_t `flexio_spi_master_handle_t::state`
- (7) `flexio_spi_shift_direction_t` `flexio_spi_master_handle_t::direction`
- (8) `flexio_spi_master_transfer_callback_t` `flexio_spi_master_handle_t::callback`
- (9) void* `flexio_spi_master_handle_t::userData`

32.7.4 Macro Definition Documentation

32.7.4.1 #define FSL_FLEXIO_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

32.7.4.2 #define FLEXIO_SPI_DUMMYDATA (0xFFFFU)

32.7.4.3 #define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

32.7.5 Typedef Documentation

32.7.5.1 typedef flexio_spi_master_handle_t flexio_spi_slave_handle_t

32.7.6 Enumeration Type Documentation

32.7.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_SPI_Busy FlexIO SPI is busy.
kStatus_FLEXIO_SPI_Idle SPI is idle.
kStatus_FLEXIO_SPI_Error FlexIO SPI error.
kStatus_FLEXIO_SPI_Timeout FlexIO SPI timeout polling status flags.

32.7.6.2 enum flexio_spi_clock_phase_t

Enumerator

kFLEXIO_SPI_ClockPhaseFirstEdge First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.
kFLEXIO_SPI_ClockPhaseSecondEdge First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

32.7.6.3 enum flexio_spi_shift_direction_t

Enumerator

kFLEXIO_SPI_MsbFirst Data transfers start with most significant bit.
kFLEXIO_SPI_LsbFirst Data transfers start with least significant bit.

32.7.6.4 enum flexio_spi_data_bitcount_mode_t

Enumerator

kFLEXIO_SPI_8BitMode 8-bit data transmission mode.
kFLEXIO_SPI_16BitMode 16-bit data transmission mode.

32.7.6.5 enum _flexio_spi_interrupt_enable

Enumerator

kFLEXIO_SPI_TxEmptyInterruptEnable Transmit buffer empty interrupt enable.
kFLEXIO_SPI_RxFullInterruptEnable Receive buffer full interrupt enable.

32.7.6.6 enum _flexio_spi_status_flags

Enumerator

kFLEXIO_SPI_TxBufferEmptyFlag Transmit buffer empty flag.
kFLEXIO_SPI_RxBufferFullFlag Receive buffer full flag.

32.7.6.7 enum _flexio_spi_dma_enable

Enumerator

kFLEXIO_SPI_TxDmaEnable Tx DMA request source.
kFLEXIO_SPI_RxDmaEnable Rx DMA request source.
kFLEXIO_SPI_DmaAllEnable All DMA request source.

32.7.6.8 enum _flexio_spi_transfer_flags

Enumerator

kFLEXIO_SPI_8bitMsb FlexIO SPI 8-bit MSB first.
kFLEXIO_SPI_8bitLsb FlexIO SPI 8-bit LSB first.
kFLEXIO_SPI_16bitMsb FlexIO SPI 16-bit MSB first.
kFLEXIO_SPI_16bitLsb FlexIO SPI 16-bit LSB first.

32.7.7 Function Documentation

32.7.7.1 void FLEXIO_SPI_MasterInit (FLEXIO_SPI_Type * *base*, flexio_spi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO_SPI_MasterGetDefaultConfig\(\)](#).

Note

1.FlexIO SPI master only support CPOL = 0, which means clock inactive low. 2.For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI master communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by 2*2=4. If FlexIO SPI master communicates with FlexIO SPI slave, the maximum baud rate is FlexIO clock frequency divided by (1.5+2.5)*2=8.

Example

```

FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_spi_master_config_t config = {
    .enableMaster = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 500000,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_MasterInit(&spiDev, &config, srcClock_Hz);

```

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>masterConfig</i>	Pointer to the flexio_spi_master_config_t structure.
<i>srcClock_Hz</i>	FlexIO source clock in Hz.

32.7.7.2 void FLEXIO_SPI_MasterDeinit (FLEXIO_SPI_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type .
-------------	--

32.7.7.3 void FLEXIO_SPI_MasterGetDefaultConfig (flexio_spi_master_config_t * *masterConfig*)

The configuration can be used directly by calling the FLEXIO_SPI_MasterConfigure(). Example:

```

flexio_spi_master_config_t masterConfig;
FLEXIO_SPI_MasterGetDefaultConfig(&masterConfig);

```

Parameters

<i>masterConfig</i>	Pointer to the flexio_spi_master_config_t structure.
---------------------	--

32.7.7.4 void FLEXIO_SPI_SlaveInit (FLEXIO_SPI_Type * *base*, flexio_spi_slave_config_t * *slaveConfig*)

The configuration structure can be filled by the user, or be set with default values by the [FLEXIO_SPI_SlaveGetDefaultConfig\(\)](#).

Note

1. Only one timer is needed in the FlexIO SPI slave. As a result, the second timer index is ignored. 2. FlexIO SPI slave only support CPOL = 0, which means clock inactive low. 3. For FlexIO SPI master, the input valid time is 1.5 clock cycles, for slave the output valid time is 2.5 clock cycles. So if FlexIO SPI slave communicates with other spi IPs, the maximum baud rate is FlexIO clock frequency divided by $3 \times 2 = 6$. If FlexIO SPI slave communicates with FlexIO SPI master, the maximum baud rate is FlexIO clock frequency divided by $(1.5 + 2.5) \times 2 = 8$. Example

```
FLEXIO_SPI_Type spiDev = {
    .flexioBase = FLEXIO,
    .SDOPinIndex = 0,
    .SDIPinIndex = 1,
    .SCKPinIndex = 2,
    .CSnPinIndex = 3,
    .shifterIndex = {0,1},
    .timerIndex = {0}
};
flexio_spi_slave_config_t config = {
    .enableSlave = true,
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .phase = kFLEXIO_SPI_ClockPhaseFirstEdge,
    .direction = kFLEXIO_SPI_MsbFirst,
    .dataMode = kFLEXIO_SPI_8BitMode
};
FLEXIO_SPI_SlaveInit(&spiDev, &config);
```

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>slaveConfig</i>	Pointer to the flexio_spi_slave_config_t structure.

32.7.7.5 void FLEXIO_SPI_SlaveDeinit (FLEXIO_SPI_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type .
-------------	--

32.7.7.6 void FLEXIO_SPI_SlaveGetDefaultConfig (flexio_spi_slave_config_t * *slaveConfig*)

The configuration can be used directly for calling the FLEXIO_SPI_SlaveConfigure(). Example:

```
flexio_spi_slave_config_t slaveConfig;
FLEXIO_SPI_SlaveGetDefaultConfig(&slaveConfig);
```

Parameters

<i>slaveConfig</i>	Pointer to the flexio_spi_slave_config_t structure.
--------------------	---

32.7.7.7 uint32_t FLEXIO_SPI_GetStatusFlags (FLEXIO_SPI_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
-------------	---

Returns

status flag; Use the status flag to AND the following flag mask and get the status.

- kFLEXIO_SPI_TxEmptyFlag
- kFLEXIO_SPI_RxEmptyFlag

32.7.7.8 void FLEXIO_SPI_ClearStatusFlags (FLEXIO_SPI_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>mask</i>	status flag The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_SPI_TxEmptyFlag • kFLEXIO_SPI_RxEmptyFlag

32.7.7.9 `void FLEXIO_SPI_EnableInterrupts (FLEXIO_SPI_Type * base, uint32_t mask)`

This function enables the FlexIO SPI interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>mask</i>	interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_SPI_RxFullInterruptEnable • kFLEXIO_SPI_TxEmptyInterruptEnable

32.7.7.10 void FLEXIO_SPI_DisableInterrupts (FLEXIO_SPI_Type * *base*, uint32_t *mask*)

This function disables the FlexIO SPI interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>mask</i>	interrupt source The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_SPI_RxFullInterruptEnable • kFLEXIO_SPI_TxEmptyInterruptEnable

32.7.7.11 void FLEXIO_SPI_EnableDMA (FLEXIO_SPI_Type * *base*, uint32_t *mask*, bool *enable*)

This function enables/disables the FlexIO SPI Tx DMA, which means that asserting the kFLEXIO_SPI_TxEmptyFlag does/doesn't trigger the DMA request.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>mask</i>	SPI DMA source.
<i>enable</i>	True means enable DMA, false means disable DMA.

32.7.7.12 static uint32_t FLEXIO_SPI_GetTxDataRegisterAddress (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*) [inline], [static]

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.

Returns

FlexIO SPI transmit data register address.

32.7.7.13 static uint32_t FLEXIO_SPI_GetRxDataRegisterAddress (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*) [inline], [static]

This function returns the SPI data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.

Returns

FlexIO SPI receive data register address.

32.7.7.14 static void FLEXIO_SPI_Enable (FLEXIO_SPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type .
<i>enable</i>	True to enable, false does not have any effect.

32.7.7.15 void FLEXIO_SPI_MasterSetBaudRate (FLEXIO_SPI_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClockHz*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>baudRate_Bps</i>	Baud Rate needed in Hz.
<i>srcClockHz</i>	SPI source clock frequency in Hz.

32.7.7.16 `static void FLEXIO_SPI_WriteData (FLEXIO_SPI_Type * base,
flexio_spi_shift_direction_t direction, uint16_t data) [inline], [static]`

Note

This is a non-blocking API, which returns directly after the data is put into the data register but the data transfer is not finished on the bus. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.
<i>data</i>	8 bit/16 bit data.

32.7.7.17 `static uint16_t FLEXIO_SPI_ReadData (FLEXIO_SPI_Type * base,
flexio_spi_shift_direction_t direction) [inline], [static]`

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.

Returns

8 bit/16 bit data received.

32.7.7.18 `status_t FLEXIO_SPI_WriteBlocking (FLEXIO_SPI_Type * base,
flexio_spi_shift_direction_t direction, const uint8_t * buffer, size_t size)`

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.
<i>buffer</i>	The data bytes to send.
<i>size</i>	The number of data bytes to send.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_FLEXIO_SPI_Timeout</i>	The transfer timed out and was aborted.

32.7.7.19 status_t FLEXIO_SPI_ReadBlocking (FLEXIO_SPI_Type * *base*, flexio_spi_shift_direction_t *direction*, uint8_t * *buffer*, size_t *size*)

Note

This function blocks using the polling method until all bytes have been received.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>direction</i>	Shift direction of MSB first or LSB first.
<i>buffer</i>	The buffer to store the received bytes.
<i>size</i>	The number of data bytes to be received.
<i>direction</i>	Shift direction of MSB first or LSB first.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_FLEXIO_SPI_Timeout</i>	The transfer timed out and was aborted.

32.7.7.20 status_t FLEXIO_SPI_MasterTransferBlocking (FLEXIO_SPI_Type * *base*, flexio_spi_transfer_t * *xfer*)

Note

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

Parameters

<i>base</i>	pointer to FLEXIO_SPI_Type structure
<i>xfer</i>	FlexIO SPI transfer structure, see flexio_spi_transfer_t .

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_FLEXIO_SPI_Timeout</i>	The transfer timed out and was aborted.

32.7.7.21 `status_t FLEXIO_SPI_MasterTransferCreateHandle (FLEXIO_SPI_Type * base, flexio_spi_master_handle_t * handle, flexio_spi_master_transfer_callback_t callback, void * userData)`

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the <code>flexio_spi_master_handle_t</code> structure to store the transfer state.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

32.7.7.22 `status_t FLEXIO_SPI_MasterTransferNonBlocking (FLEXIO_SPI_Type * base, flexio_spi_master_handle_t * handle, flexio_spi_transfer_t * xfer)`

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.
<i>xfer</i>	FlexIO SPI transfer structure. See flexio_spi_transfer_t .

Return values

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

32.7.7.23 void FLEXIO_SPI_MasterTransferAbort (FLEXIO_SPI_Type * *base*, flexio_spi_master_handle_t * *handle*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

32.7.7.24 status_t FLEXIO_SPI_MasterTransferGetCount (FLEXIO_SPI_Type * *base*, flexio_spi_master_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_master_handle_t structure to store 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.

32.7.7.25 void FLEXIO_SPI_MasterTransferHandleIRQ (void * *spiType*, void * *spiHandle*)

Parameters

<i>spiType</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>spiHandle</i>	Pointer to the flexio_spi_master_handle_t structure to store the transfer state.

32.7.7.26 `status_t FLEXIO_SPI_SlaveTransferCreateHandle (FLEXIO_SPI_Type * base, flexio_spi_slave_handle_t * handle, flexio_spi_slave_transfer_callback_t callback, void * userData)`

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

32.7.7.27 `status_t FLEXIO_SPI_SlaveTransferNonBlocking (FLEXIO_SPI_Type * base, flexio_spi_slave_handle_t * handle, flexio_spi_transfer_t * xfer)`

This function sends data using IRQ. This is a non-blocking function, which returns right away. When all data is sent out/received, the callback function is called.

Parameters

<i>handle</i>	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.
<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>xfer</i>	FlexIO SPI transfer structure. See flexio_spi_transfer_t .

Return values

<i>kStatus_Success</i>	Successfully start a transfer.
<i>kStatus_InvalidArgument</i>	Input argument is invalid.
<i>kStatus_FLEXIO_SPI_Busy</i>	SPI is not idle; it is running another transfer.

32.7.7.28 static void FLEXIO_SPI_SlaveTransferAbort (FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

32.7.7.29 static status_t FLEXIO_SPI_SlaveTransferGetCount (FLEXIO_SPI_Type * *base*, flexio_spi_slave_handle_t * *handle*, size_t * *count*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to the flexio_spi_slave_handle_t structure to store 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.

32.7.7.30 void FLEXIO_SPI_SlaveTransferHandleIRQ (void * *spiType*, void * *spiHandle*)

Parameters

<i>spiType</i>	Pointer to the FLEXIO_SPI_Type structure.
<i>spiHandle</i>	Pointer to the flexio_spi_slave_handle_t structure to store the transfer state.

32.7.8 FlexIO eDMA SPI Driver

32.7.8.1 Overview

Data Structures

- struct `flexio_spi_master_edma_handle_t`
FlexIO SPI eDMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef
`flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t`
Slave handle is the same with master handle.
- typedef void(* `flexio_spi_master_edma_transfer_callback_t`)(`FLEXIO_SPI_Type` *base, `flexio_spi_master_edma_handle_t` *handle, `status_t` status, void *userData)
FlexIO SPI master callback for finished transmit.
- typedef void(* `flexio_spi_slave_edma_transfer_callback_t`)(`FLEXIO_SPI_Type` *base, `flexio_spi_slave_edma_handle_t` *handle, `status_t` status, void *userData)
FlexIO SPI slave callback for finished transmit.

Driver version

- #define `FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 2, 1))
FlexIO SPI EDMA driver version.

eDMA Transactional

- `status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA` (`FLEXIO_SPI_Type` *base, `flexio_spi_master_edma_handle_t` *handle, `flexio_spi_master_edma_transfer_callback_t` callback, void *userData, `edma_handle_t` *txHandle, `edma_handle_t` *rxHandle)
Initializes the FlexIO SPI master eDMA handle.
- `status_t FLEXIO_SPI_MasterTransferEDMA` (`FLEXIO_SPI_Type` *base, `flexio_spi_master_edma_handle_t` *handle, `flexio_spi_transfer_t` *xfer)
Performs a non-blocking FlexIO SPI transfer using eDMA.
- void `FLEXIO_SPI_MasterTransferAbortEDMA` (`FLEXIO_SPI_Type` *base, `flexio_spi_master_edma_handle_t` *handle)
Aborts a FlexIO SPI transfer using eDMA.
- `status_t FLEXIO_SPI_MasterTransferGetCountEDMA` (`FLEXIO_SPI_Type` *base, `flexio_spi_master_edma_handle_t` *handle, `size_t` *count)
Gets the number of bytes transferred so far using FlexIO SPI master eDMA.
- static void `FLEXIO_SPI_SlaveTransferCreateHandleEDMA` (`FLEXIO_SPI_Type` *base, `flexio_spi_slave_edma_handle_t` *handle, `flexio_spi_slave_edma_transfer_callback_t` callback, void *userData, `edma_handle_t` *txHandle, `edma_handle_t` *rxHandle)
Initializes the FlexIO SPI slave eDMA handle.

- `status_t FLEXIO_SPI_SlaveTransferEDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle, flexio_spi_transfer_t *xfer)`
Performs a non-blocking FlexIO SPI transfer using eDMA.
- `static void FLEXIO_SPI_SlaveTransferAbortEDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle)`
Aborts a FlexIO SPI transfer using eDMA.
- `static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA (FLEXIO_SPI_Type *base, flexio_spi_slave_edma_handle_t *handle, size_t *count)`
Gets the number of bytes transferred so far using FlexIO SPI slave eDMA.

32.7.8.2 Data Structure Documentation

32.7.8.2.1 struct flexio_spi_master_edma_handle

typedef for flexio_spi_master_edma_handle_t in advance.

Data Fields

- `size_t transferSize`
Total bytes to be transferred.
- `uint8_t nbytes`
eDMA minor byte transfer count initially configured.
- `bool txInProgress`
Send transfer in progress.
- `bool rxInProgress`
Receive transfer in progress.
- `edma_handle_t * txHandle`
DMA handler for SPI send.
- `edma_handle_t * rxHandle`
DMA handler for SPI receive.
- `flexio_spi_master_edma_transfer_callback_t callback`
Callback for SPI DMA transfer.
- `void * userData`
User Data for SPI DMA callback.

Field Documentation

(1) `size_t flexio_spi_master_edma_handle_t::transferSize`

(2) `uint8_t flexio_spi_master_edma_handle_t::nbytes`

32.7.8.3 Macro Definition Documentation

32.7.8.3.1 `#define FSL_FLEXIO_SPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))`

32.7.8.4 Typedef Documentation

32.7.8.4.1 typedef flexio_spi_master_edma_handle_t flexio_spi_slave_edma_handle_t

32.7.8.5 Function Documentation

32.7.8.5.1 status_t FLEXIO_SPI_MasterTransferCreateHandleEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*, flexio_spi_master_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *txHandle*, edma_handle_t * *rxHandle*)

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

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
<i>callback</i>	SPI callback, NULL means no callback.
<i>userData</i>	callback function parameter.
<i>txHandle</i>	User requested eDMA handle for FlexIO SPI RX eDMA transfer.
<i>rxHandle</i>	User requested eDMA handle for FlexIO SPI TX eDMA transfer.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO SPI eDMA type/handle table out of range.

32.7.8.5.2 status_t FLEXIO_SPI_MasterTransferEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*, flexio_spi_transfer_t * *xfer*)

Note

This interface returns immediately after transfer initiates. Call FLEXIO_SPI_MasterGetTransferCountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
-------------	---

<i>handle</i>	Pointer to flexio_spi_master_edma_handle_t structure to store the transfer state.
<i>xfer</i>	Pointer to FlexIO SPI transfer structure.

Return values

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

32.7.8.5.3 void FLEXIO_SPI_MasterTransferAbortEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*)

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	FlexIO SPI eDMA handle pointer.

32.7.8.5.4 status_t FLEXIO_SPI_MasterTransferGetCountEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_master_edma_handle_t * *handle*, size_t * *count*)

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	FlexIO SPI eDMA handle pointer.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

32.7.8.5.5 static void FLEXIO_SPI_SlaveTransferCreateHandleEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle*, flexio_spi_slave_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *txHandle*, edma_handle_t * *rxHandle*) [inline], [static]

This function initializes the FlexIO SPI slave eDMA handle.

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
<i>callback</i>	SPI callback, NULL means no callback.
<i>userData</i>	callback function parameter.
<i>txHandle</i>	User requested eDMA handle for FlexIO SPI TX eDMA transfer.
<i>rxHandle</i>	User requested eDMA handle for FlexIO SPI RX eDMA transfer.

32.7.8.5.6 status_t FLEXIO_SPI_SlaveTransferEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle*, flexio_spi_transfer_t * *xfer*)

Note

This interface returns immediately after transfer initiates. Call FLEXIO_SPI_SlaveGetTransfer-CountEDMA to poll the transfer status and check whether the FlexIO SPI transfer is finished.

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.
<i>xfer</i>	Pointer to FlexIO SPI transfer structure.

Return values

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

32.7.8.5.7 static void FLEXIO_SPI_SlaveTransferAbortEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle*) [inline], [static]

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	Pointer to flexio_spi_slave_edma_handle_t structure to store the transfer state.

32.7.8.5.8 static status_t FLEXIO_SPI_SlaveTransferGetCountEDMA (FLEXIO_SPI_Type * *base*, flexio_spi_slave_edma_handle_t * *handle*, size_t * *count*) [inline], [static]

Parameters

<i>base</i>	Pointer to FLEXIO_SPI_Type structure.
<i>handle</i>	FlexIO SPI eDMA handle pointer.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

32.8 FlexIO UART Driver

32.8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Universal Asynchronous Receiver/Transmitter (UART) function using the Flexible I/O.

FlexIO UART driver includes functional APIs and transactional APIs. Functional APIs target low-level APIs. Functional APIs can be used for the FlexIO UART initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the FlexIO UART peripheral and how to organize functional APIs to meet the application requirements. All functional API use the `FLEXIO_UART_Type *` as the first parameter. FlexIO UART functional operation groups provide the functional APIs set.

Transactional APIs target high-level APIs. Transactional APIs can be used to enable the peripheral and also in the application if the code size and performance of transactional APIs satisfy requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `flexio_uart_handle_t` as the second parameter. Initialize the handle by calling the `FLEXIO_UART_TransferCreateHandle()` API.

Transactional APIs support asynchronous transfer. This means that the functions `FLEXIO_UART_SendNonBlocking()` and `FLEXIO_UART_ReceiveNonBlocking()` set up an interrupt for data transfer. When the transfer is complete, the upper layer is notified through a callback function with the `kStatus_FLEXIO_UART_TxIdle` and `kStatus_FLEXIO_UART_RxIdle` status.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size through calling the `FLEXIO_UART_InstallRingBuffer()`. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The function `FLEXIO_UART_ReceiveNonBlocking()` first gets data from the ring buffer. If ring buffer does not have enough data, the function returns the data to the ring buffer and saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `kStatus_FLEXIO_UART_RxIdle` status.

If the receive ring buffer is full, the upper layer is informed through a callback with status `kStatus_FLEXIO_UART_RxRingBufferOverflow`. In the callback function, the upper layer reads data from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when calling the `FLEXIO_UART_InstallRingBuffer`. Note that one byte is reserved for the ring buffer maintenance. Create a handle as follows.

```
FLEXIO_UART_InstallRingBuffer(&uartDev, &handle, &ringBuffer, 32);
```

In this example, the buffer size is 32. However, only 31 bytes are used for saving data.

32.8.2 Typical use case

32.8.2.1 FlexIO UART send/receive using a polling method

```
uint8_t ch;
```



```

FLEXIO_UART_Type uartDev;
status_t result = kStatus_Success;
flexio_uart_user_config user_config;
FLEXIO_UART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableUart = true;

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
    return;
}
FLEXIO_UART_WriteBlocking(&uartDev, txbuff, sizeof(txbuff));

while(1)
{
    FLEXIO_UART_ReadBlocking(&uartDev, &ch, 1);
    FLEXIO_UART_WriteBlocking(&uartDev, &ch, 1);
}

```

32.8.2.2 FlexIO UART send/receive using an interrupt method

```

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

```

```

uartDev.flexioBase = BOARD_FLEXIO_BASE;
uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
uartDev.shifterIndex[0] = 0U;
uartDev.shifterIndex[1] = 1U;
uartDev.timerIndex[0] = 0U;
uartDev.timerIndex[1] = 1U;

result = FLEXIO_UART_Init(&uartDev, &user_config, 12000000U);
//Check if configuration is correct.
if(result != kStatus_Success)
{
    return;
}

FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
    FLEXIO_UART_UserCallback, NULL);

// Prepares to send.
sendXfer.data = sendData;
sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
txFinished = false;

// Sends out.
FLEXIO_UART_SendNonBlocking(&uartDev, &g_uartHandle, &sendXfer);

// Send finished.
while (!txFinished)
{
}

// Prepares to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
rxFinished = false;

// Receives.
FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, NULL);

// Receive finished.
while (!rxFinished)
{
}

// ...
}

```

32.8.2.3 FlexIO UART receive using the ringbuffer feature

```

#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE 32

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{

```

```

    userData = userData;

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    size_t bytesRead;
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;

    result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    FLEXIO_UART_TransferCreateHandle(&uartDev, &g_uartHandle,
        FLEXIO_UART_UserCallback, NULL);
    FLEXIO_UART_InstallRingBuffer(&uartDev, &g_uartHandle, ringBuffer, RING_BUFFER_SIZE);

    // Receive is working in the background to the ring buffer.

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = RX_DATA_SIZE;
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveNonBlocking(&uartDev, &g_uartHandle, &receiveXfer, &bytesRead);

    if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
    {
        ;
    }
    else
    {
        if (bytesRead) /* Received some data, process first. */
        {
            ;
        }

        // Receive finished.
        while (!rxFinished)
        {
        }
    }

    // ...
}

```

32.8.2.4 FlexIO UART send/receive using a DMA method

```

FLEXIO_UART_Type uartDev;
flexio_uart_handle_t g_uartHandle;
dma_handle_t g_uartTxDmaHandle;
dma_handle_t g_uartRxDmaHandle;
flexio_uart_config_t user_config;
flexio_uart_transfer_t sendXfer;
flexio_uart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void FLEXIO_UART_UserCallback(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle,
    status_t status, void *userData)
{
    userData = userData;

    if (kStatus_FLEXIO_UART_TxIdle == status)
    {
        txFinished = true;
    }

    if (kStatus_FLEXIO_UART_RxIdle == status)
    {
        rxFinished = true;
    }
}

void main(void)
{
    //...

    FLEXIO_UART_GetDefaultConfig(&user_config);
    user_config.baudRate_Bps = 115200U;
    user_config.enableUart = true;

    uartDev.flexioBase = BOARD_FLEXIO_BASE;
    uartDev.TxPinIndex = FLEXIO_UART_TX_PIN;
    uartDev.RxPinIndex = FLEXIO_UART_RX_PIN;
    uartDev.shifterIndex[0] = 0U;
    uartDev.shifterIndex[1] = 1U;
    uartDev.timerIndex[0] = 0U;
    uartDev.timerIndex[1] = 1U;
    result = FLEXIO_UART_Init(&uartDev, &user_config, 48000000U);
    //Check if configuration is correct.
    if(result != kStatus_Success)
    {
        return;
    }

    /* Init DMAMUX. */
    DMAMUX_Init(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR)

    /* Init the DMA/EDMA module */
#ifdef FSL_FEATURE_SOC_DMA_COUNT && FSL_FEATURE_SOC_DMA_COUNT > 0U
    DMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR);
    DMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL);
    DMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_DMA_COUNT */

#ifdef FSL_FEATURE_SOC_EDMA_COUNT && FSL_FEATURE_SOC_EDMA_COUNT > 0U
    edma_config_t edmaConfig;

    EDMA_GetDefaultConfig(&edmaConfig);
    EDMA_Init(EXAMPLE_FLEXIO_UART_DMA_BASEADDR, &edmaConfig);

```

```

    EDMA_CreateHandle(&g_uartTxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
FLEXIO_UART_TX_DMA_CHANNEL);
    EDMA_CreateHandle(&g_uartRxDmaHandle, EXAMPLE_FLEXIO_UART_DMA_BASEADDR,
FLEXIO_UART_RX_DMA_CHANNEL);
#endif /* FSL_FEATURE_SOC_EDMA_COUNT */

    dma_request_source_tx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
shifterIndex[0]);
    dma_request_source_rx = (dma_request_source_t)(FLEXIO_DMA_REQUEST_BASE + uartDev.
shifterIndex[1]);

    /* Requests DMA channels for transmit and receive. */
    DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_TX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_tx);
    DMAMUX_SetSource(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR, FLEXIO_UART_RX_DMA_CHANNEL, (
dma_request_source_t)dma_request_source_rx);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
FLEXIO_UART_TX_DMA_CHANNEL);
    DMAMUX_EnableChannel(EXAMPLE_FLEXIO_UART_DMAMUX_BASEADDR,
FLEXIO_UART_RX_DMA_CHANNEL);

    FLEXIO_UART_TransferCreateHandleDMA(&uartDev, &g_uartHandle, FLEXIO_UART_UserCallback, NULL, &
g_uartTxDmaHandle, &g_uartRxDmaHandle);

    // Prepares to send.
    sendXfer.data = sendData
    sendXfer.dataSize = sizeof(sendData)/sizeof(sendData[0]);
    txFinished = false;

    // Sends out.
    FLEXIO_UART_SendDMA(&uartDev, &g_uartHandle, &sendXfer);

    // Send finished.
    while (!txFinished)
    {
    }

    // Prepares to receive.
    receiveXfer.data = receiveData;
    receiveXfer.dataSize = sizeof(receiveData)/sizeof(receiveData[0]);
    rxFinished = false;

    // Receives.
    FLEXIO_UART_ReceiveDMA(&uartDev, &g_uartHandle, &receiveXfer, NULL);

    // Receive finished.
    while (!rxFinished)
    {
    }

    // ...
}

```

Modules

- [FlexIO eDMA UART Driver](#)

Data Structures

- struct [FLEXIO_UART_Type](#)
Define FlexIO UART access structure typedef. [More...](#)

- struct `flexio_uart_config_t`
Define FlexIO UART user configuration structure. [More...](#)
- struct `flexio_uart_transfer_t`
Define FlexIO UART transfer structure. [More...](#)
- struct `flexio_uart_handle_t`
Define FLEXIO UART handle structure. [More...](#)

Macros

- #define `UART_RETRY_TIMES` 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */
Retry times for waiting flag.

Typedefs

- typedef void(* `flexio_uart_transfer_callback_t`)(`FLEXIO_UART_Type` *base, `flexio_uart_handle_t` *handle, `status_t` status, void *userData)
FlexIO UART transfer callback function.

Enumerations

- enum {
`kStatus_FLEXIO_UART_TxBusy` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 0),
`kStatus_FLEXIO_UART_RxBusy` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 1),
`kStatus_FLEXIO_UART_TxIdle` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 2),
`kStatus_FLEXIO_UART_RxIdle` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 3),
`kStatus_FLEXIO_UART_ERROR` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 4),
`kStatus_FLEXIO_UART_RxRingBufferOverrun`,
`kStatus_FLEXIO_UART_RxHardwareOverrun` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 6),
`kStatus_FLEXIO_UART_Timeout` = MAKE_STATUS(kStatusGroup_FLEXIO_UART, 7),
`kStatus_FLEXIO_UART_BaudrateNotSupport` }
Error codes for the UART driver.
- enum `flexio_uart_bit_count_per_char_t` {
`kFLEXIO_UART_7BitsPerChar` = 7U,
`kFLEXIO_UART_8BitsPerChar` = 8U,
`kFLEXIO_UART_9BitsPerChar` = 9U }
FlexIO UART bit count per char.
- enum `_flexio_uart_interrupt_enable` {
`kFLEXIO_UART_TxDataRegEmptyInterruptEnable` = 0x1U,
`kFLEXIO_UART_RxDataRegFullInterruptEnable` = 0x2U }
Define FlexIO UART interrupt mask.
- enum `_flexio_uart_status_flags` {

```
kFLEXIO_UART_TxDataRegEmptyFlag = 0x1U,
kFLEXIO_UART_RxDataRegFullFlag = 0x2U,
kFLEXIO_UART_RxOverRunFlag = 0x4U }
```

Define FlexIO UART status mask.

Driver version

- #define **FSL_FLEXIO_UART_DRIVER_VERSION** (**MAKE_VERSION**(2, 4, 0))
FlexIO UART driver version.

Initialization and deinitialization

- **status_t FLEXIO_UART_Init** (**FLEXIO_UART_Type** *base, const **flexio_uart_config_t** *userConfig, **uint32_t** srcClock_Hz)
Ungates the FlexIO clock, resets the FlexIO module, configures FlexIO UART hardware, and configures the FlexIO UART with FlexIO UART configuration.
- **void FLEXIO_UART_Deinit** (**FLEXIO_UART_Type** *base)
Resets the FlexIO UART shifter and timer config.
- **void FLEXIO_UART_GetDefaultConfig** (**flexio_uart_config_t** *userConfig)
Gets the default configuration to configure the FlexIO UART.

Status

- **uint32_t FLEXIO_UART_GetStatusFlags** (**FLEXIO_UART_Type** *base)
Gets the FlexIO UART status flags.
- **void FLEXIO_UART_ClearStatusFlags** (**FLEXIO_UART_Type** *base, **uint32_t** mask)
Gets the FlexIO UART status flags.

Interrupts

- **void FLEXIO_UART_EnableInterrupts** (**FLEXIO_UART_Type** *base, **uint32_t** mask)
Enables the FlexIO UART interrupt.
- **void FLEXIO_UART_DisableInterrupts** (**FLEXIO_UART_Type** *base, **uint32_t** mask)
Disables the FlexIO UART interrupt.

DMA Control

- **static uint32_t FLEXIO_UART_GetTxDataRegisterAddress** (**FLEXIO_UART_Type** *base)
Gets the FlexIO UART transmit data register address.
- **static uint32_t FLEXIO_UART_GetRxDataRegisterAddress** (**FLEXIO_UART_Type** *base)
Gets the FlexIO UART receive data register address.
- **static void FLEXIO_UART_EnableTxDMA** (**FLEXIO_UART_Type** *base, **bool** enable)
Enables/disables the FlexIO UART transmit DMA.
- **static void FLEXIO_UART_EnableRxDMA** (**FLEXIO_UART_Type** *base, **bool** enable)

Enables/disables the FlexIO UART receive DMA.

Bus Operations

- static void `FLEXIO_UART_Enable` (`FLEXIO_UART_Type` *base, bool enable)
Enables/disables the FlexIO UART module operation.
- static void `FLEXIO_UART_WriteByte` (`FLEXIO_UART_Type` *base, const uint8_t *buffer)
Writes one byte of data.
- static void `FLEXIO_UART_ReadByte` (`FLEXIO_UART_Type` *base, uint8_t *buffer)
Reads one byte of data.
- `status_t` `FLEXIO_UART_WriteBlocking` (`FLEXIO_UART_Type` *base, const uint8_t *txData, size_t txSize)
Sends a buffer of data bytes.
- `status_t` `FLEXIO_UART_ReadBlocking` (`FLEXIO_UART_Type` *base, uint8_t *rxData, size_t rxSize)
Receives a buffer of bytes.

Transactional

- `status_t` `FLEXIO_UART_TransferCreateHandle` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle, flexio_uart_transfer_callback_t callback, void *userData)
Initializes the UART handle.
- void `FLEXIO_UART_TransferStartRingBuffer` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle, uint8_t *ringBuffer, size_t ringBufferSize)
Sets up the RX ring buffer.
- void `FLEXIO_UART_TransferStopRingBuffer` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle)
Aborts the background transfer and uninstalls the ring buffer.
- `status_t` `FLEXIO_UART_TransferSendNonBlocking` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer)
Transmits a buffer of data using the interrupt method.
- void `FLEXIO_UART_TransferAbortSend` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle)
Aborts the interrupt-driven data transmit.
- `status_t` `FLEXIO_UART_TransferGetSendCount` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle, size_t *count)
Gets the number of bytes sent.
- `status_t` `FLEXIO_UART_TransferReceiveNonBlocking` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle, flexio_uart_transfer_t *xfer, size_t *receivedBytes)
Receives a buffer of data using the interrupt method.
- void `FLEXIO_UART_TransferAbortReceive` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle)
Aborts the receive data which was using IRQ.
- `status_t` `FLEXIO_UART_TransferGetReceiveCount` (`FLEXIO_UART_Type` *base, flexio_uart_handle_t *handle, size_t *count)
Gets the number of bytes received.
- void `FLEXIO_UART_TransferHandleIRQ` (void *uartType, void *uartHandle)

FlexIO UART IRQ handler function.

32.8.3 Data Structure Documentation

32.8.3.1 struct FLEXIO_UART_Type

Data Fields

- FLEXIO_Type * [flexioBase](#)
FlexIO base pointer.
- uint8_t [TxPinIndex](#)
Pin select for UART_Tx.
- uint8_t [RxPinIndex](#)
Pin select for UART_Rx.
- uint8_t [shifterIndex](#) [2]
Shifter index used in FlexIO UART.
- uint8_t [timerIndex](#) [2]
Timer index used in FlexIO UART.

Field Documentation

- (1) FLEXIO_Type* FLEXIO_UART_Type::flexioBase
- (2) uint8_t FLEXIO_UART_Type::TxPinIndex
- (3) uint8_t FLEXIO_UART_Type::RxPinIndex
- (4) uint8_t FLEXIO_UART_Type::shifterIndex[2]
- (5) uint8_t FLEXIO_UART_Type::timerIndex[2]

32.8.3.2 struct flexio_uart_config_t

Data Fields

- bool [enableUart](#)
Enable/disable FlexIO UART TX & RX.
- bool [enableInDoze](#)
Enable/disable FlexIO operation in doze mode.
- bool [enableInDebug](#)
Enable/disable FlexIO operation in debug mode.
- bool [enableFastAccess](#)
*Enable/disable fast access to FlexIO registers,
fast access requires the FlexIO clock to be at least twice the frequency of the bus clock.*
- uint32_t [baudRate_Bps](#)
Baud rate in Bps.
- [flexio_uart_bit_count_per_char_t](#) bitCountPerChar
number of bits, 7/8/9 -bit

Field Documentation

- (1) `bool flexio_uart_config_t::enableUart`
- (2) `bool flexio_uart_config_t::enableFastAccess`
- (3) `uint32_t flexio_uart_config_t::baudRate_Bps`

32.8.3.3 struct flexio_uart_transfer_t

Data Fields

- `size_t dataSize`
Transfer size.
- `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* flexio_uart_transfer_t::data`
- (2) `uint8_t* flexio_uart_transfer_t::rxData`
- (3) `const uint8_t* flexio_uart_transfer_t::txData`

32.8.3.4 struct _flexio_uart_handle

Data Fields

- `const uint8_t *volatile txData`
Address of remaining data to send.
- `volatile size_t txDataSize`
Size of the remaining data to send.
- `uint8_t *volatile rxData`
Address of remaining data to receive.
- `volatile size_t rxDataSize`
Size of the remaining data to receive.
- `size_t txDataSizeAll`
Total bytes to be sent.
- `size_t rxDataSizeAll`
Total bytes to be received.
- `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.
- `flexio_uart_transfer_callback_t` `callback`
Callback function.
- void * `userData`
UART callback function parameter.
- volatile uint8_t `txState`
TX transfer state.
- volatile uint8_t `rxState`
RX transfer state.

Field Documentation

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

32.8.4 Macro Definition Documentation

32.8.4.1 `#define FSL_FLEXIO_UART_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))`

32.8.4.2 `#define UART_RETRY_TIMES 0U` /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */

32.8.5 Typedef Documentation

32.8.5.1 `typedef void(* flexio_uart_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_handle_t *handle, status_t status, void *userData)`

32.8.6 Enumeration Type Documentation

32.8.6.1 anonymous enum

Enumerator

kStatus_FLEXIO_UART_TxBusy Transmitter is busy.
kStatus_FLEXIO_UART_RxBusy Receiver is busy.
kStatus_FLEXIO_UART_TxIdle UART transmitter is idle.
kStatus_FLEXIO_UART_RxIdle UART receiver is idle.
kStatus_FLEXIO_UART_ERROR ERROR happens on UART.
kStatus_FLEXIO_UART_RxRingBufferOverrun UART RX software ring buffer overrun.
kStatus_FLEXIO_UART_RxHardwareOverrun UART RX receiver overrun.
kStatus_FLEXIO_UART_Timeout UART times out.
kStatus_FLEXIO_UART_BaudrateNotSupport Baudrate is not supported in current clock source.

32.8.6.2 enum flexio_uart_bit_count_per_char_t

Enumerator

kFLEXIO_UART_7BitsPerChar 7-bit data characters
kFLEXIO_UART_8BitsPerChar 8-bit data characters
kFLEXIO_UART_9BitsPerChar 9-bit data characters

32.8.6.3 enum _flexio_uart_interrupt_enable

Enumerator

kFLEXIO_UART_TxDataRegEmptyInterruptEnable Transmit buffer empty interrupt enable.
kFLEXIO_UART_RxDataRegFullInterruptEnable Receive buffer full interrupt enable.

32.8.6.4 enum _flexio_uart_status_flags

Enumerator

kFLEXIO_UART_TxDataRegEmptyFlag Transmit buffer empty flag.
kFLEXIO_UART_RxDataRegFullFlag Receive buffer full flag.
kFLEXIO_UART_RxOverRunFlag Receive buffer over run flag.

32.8.7 Function Documentation

32.8.7.1 `status_t FLEXIO_UART_Init (FLEXIO_UART_Type * base, const flexio_uart_config_t * userConfig, uint32_t srcClock_Hz)`

The configuration structure can be filled by the user or be set with default values by [FLEXIO_UART_GetDefaultConfig\(\)](#).

Example

```
FLEXIO_UART_Type base = {
    .flexioBase = FLEXIO,
    .TxPinIndex = 0,
    .RxPinIndex = 1,
    .shifterIndex = {0,1},
    .timerIndex = {0,1}
};
flexio_uart_config_t config = {
    .enableInDoze = false,
    .enableInDebug = true,
    .enableFastAccess = false,
    .baudRate_Bps = 115200U,
    .bitCountPerChar = 8
};
FLEXIO_UART_Init(base, &config, srcClock_Hz);
```

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>userConfig</i>	Pointer to the flexio_uart_config_t structure.
<i>srcClock_Hz</i>	FlexIO source clock in Hz.

Return values

<i>kStatus_Success</i>	Configuration success.
<i>kStatus_FLEXIO_UART-BaudrateNotSupport</i>	Baudrate is not supported for current clock source frequency.

32.8.7.2 `void FLEXIO_UART_Deinit (FLEXIO_UART_Type * base)`

Note

After calling this API, call the `FLEXIO_UART_Init` to use the FlexIO UART module.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type structure
-------------	---

32.8.7.3 void FLEXIO_UART_GetDefaultConfig (flexio_uart_config_t * *userConfig*)

The configuration can be used directly for calling the [FLEXIO_UART_Init\(\)](#). Example:

```
flexio_uart_config_t config;
FLEXIO_UART_GetDefaultConfig(&userConfig);
```

Parameters

<i>userConfig</i>	Pointer to the flexio_uart_config_t structure.
-------------------	--

32.8.7.4 uint32_t FLEXIO_UART_GetStatusFlags (FLEXIO_UART_Type * *base*)

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
-------------	--

Returns

FlexIO UART status flags.

32.8.7.5 void FLEXIO_UART_ClearStatusFlags (FLEXIO_UART_Type * *base*, uint32_t *mask*)

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>mask</i>	Status flag. The parameter can be any combination of the following values: <ul style="list-style-type: none"> • kFLEXIO_UART_TxDataRegEmptyFlag • kFLEXIO_UART_RxEmptyFlag • kFLEXIO_UART_RxOverRunFlag

32.8.7.6 void FLEXIO_UART_EnableInterrupts (FLEXIO_UART_Type * *base*, uint32_t *mask*)

This function enables the FlexIO UART interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>mask</i>	Interrupt source.

32.8.7.7 void FLEXIO_UART_DisableInterrupts (FLEXIO_UART_Type * *base*, uint32_t *mask*)

This function disables the FlexIO UART interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>mask</i>	Interrupt source.

32.8.7.8 static uint32_t FLEXIO_UART_GetTxDataRegisterAddress (FLEXIO_UART_Type * *base*) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
-------------	--

Returns

FlexIO UART transmit data register address.

32.8.7.9 static uint32_t FLEXIO_UART_GetRxDataRegisterAddress (FLEXIO_UART_Type * *base*) [inline], [static]

This function returns the UART data register address, which is mainly used by DMA/eDMA.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
-------------	--

Returns

FlexIO UART receive data register address.

32.8.7.10 static void FLEXIO_UART_EnableTxDMA (FLEXIO_UART_Type * *base*, bool *enable*) [inline], [static]

This function enables/disables the FlexIO UART Tx DMA, which means asserting the kFLEXIO_UART_TxDataRegEmptyFlag does/doesn't trigger the DMA request.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>enable</i>	True to enable, false to disable.

32.8.7.11 static void FLEXIO_UART_EnableRxDMA (FLEXIO_UART_Type * *base*, bool *enable*) [inline], [static]

This function enables/disables the FlexIO UART Rx DMA, which means asserting kFLEXIO_UART_RxDataRegFullFlag does/doesn't trigger the DMA request.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>enable</i>	True to enable, false to disable.

32.8.7.12 static void FLEXIO_UART_Enable (FLEXIO_UART_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type .
<i>enable</i>	True to enable, false does not have any effect.

32.8.7.13 static void FLEXIO_UART_WriteByte (FLEXIO_UART_Type * *base*, const uint8_t * *buffer*) [inline], [static]

Note

This is a non-blocking API, which returns directly after the data is put into the data register. Ensure that the TxEmptyFlag is asserted before calling this API.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
-------------	--

<i>buffer</i>	The data bytes to send.
---------------	-------------------------

32.8.7.14 `static void FLEXIO_UART_ReadByte (FLEXIO_UART_Type * base, uint8_t * buffer) [inline], [static]`

Note

This is a non-blocking API, which returns directly after the data is read from the data register. Ensure that the RxFullFlag is asserted before calling this API.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>buffer</i>	The buffer to store the received bytes.

32.8.7.15 `status_t FLEXIO_UART_WriteBlocking (FLEXIO_UART_Type * base, const uint8_t * txData, size_t txSize)`

Note

This function blocks using the polling method until all bytes have been sent.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>txData</i>	The data bytes to send.
<i>txSize</i>	The number of data bytes to send.

Return values

<i>kStatus_FLEXIO_UART- _Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully wrote all data.

32.8.7.16 `status_t FLEXIO_UART_ReadBlocking (FLEXIO_UART_Type * base, uint8_t * rxData, size_t rxSize)`

Note

This function blocks using the polling method until all bytes have been received.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>rxData</i>	The buffer to store the received bytes.
<i>rxSize</i>	The number of data bytes to be received.

Return values

<i>kStatus_FLEXIO_UART- _Timeout</i>	Transmission timed out and was aborted.
<i>kStatus_Success</i>	Successfully received all data.

32.8.7.17 status_t FLEXIO_UART_TransferCreateHandle (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_callback_t *callback*, void * *userData*)

This function initializes the FlexIO UART handle, which can be used for other FlexIO UART transactional APIs. Call this API once to get the initialized handle.

The UART driver supports the "background" receiving, which means that users can set up a RX ring buffer optionally. Data received is stored into the ring buffer even when the user doesn't call the [FLEXIO_UART_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly. The ring buffer is disabled if passing NULL as *ringBuffer*.

Parameters

<i>base</i>	to FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO type/handle/ISR table out of range.

32.8.7.18 void FLEXIO_UART_TransferStartRingBuffer (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, uint8_t * *ringBuffer*, size_t *ringBufferSize*)

This function sets up the RX ring buffer to a specific UART handle.

When the RX ring buffer is used, data received is stored into the ring buffer even when the user doesn't

call the `UART_ReceiveNonBlocking()` API. If there is already data received in the ring buffer, users can get the received data from the ring buffer directly.

Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, only 31 bytes are used for saving data.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.
<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.

32.8.7.19 void FLEXIO_UART_TransferStopRingBuffer (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*)

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

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the <code>flexio_uart_handle_t</code> structure to store the transfer state.

32.8.7.20 status_t FLEXIO_UART_TransferSendNonBlocking (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_t * *xfer*)

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

Note

The `kStatus_FLEXIO_UART_TxIdle` is passed to the upper layer when all data is written to the TX register. However, it does not ensure that all data is sent out.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>xfer</i>	FlexIO UART transfer structure. See flexio_uart_transfer_t .

Return values

<i>kStatus_Success</i>	Successfully starts the data transmission.
<i>kStatus_UART_TxBusy</i>	Previous transmission still not finished, data not written to the TX register.

32.8.7.21 void FLEXIO_UART_TransferAbortSend (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*)

This function aborts the interrupt-driven data sending. Get the remainBytes to find out how many bytes are still not sent out.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.

32.8.7.22 status_t FLEXIO_UART_TransferGetSendCount (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, size_t * *count*)

This function gets the number of bytes sent driven by interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

Return values

<i>kStatus_NoTransferInProgress</i>	transfer has finished or no transfer in progress.
<i>kStatus_Success</i>	Successfully return the count.

32.8.7.23 **status_t FLEXIO_UART_TransferReceiveNonBlocking (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*, flexio_uart_transfer_t * *xfer*, size_t * *receivedBytes*)**

This function receives data using the interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in ring buffer is not enough to read, the receive request is saved by the UART driver. When new data arrives, the receive request is serviced first. When all data is received, the UART driver notifies the upper layer through a callback function and passes the status parameter *kStatus_FLEXIO_UART_RxIdle*. For example, if the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer, the 5 bytes are copied to *xfer->data*. This function returns with the parameter *receivedBytes* set to 5. For the last 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the UART driver notifies 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>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the <i>flexio_uart_handle_t</i> structure to store the transfer state.
<i>xfer</i>	UART transfer structure. See flexio_uart_transfer_t .
<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_FLEXIO_UART_RxBusy</i>	Previous receive request is not finished.

32.8.7.24 **void FLEXIO_UART_TransferAbortReceive (FLEXIO_UART_Type * *base*, flexio_uart_handle_t * *handle*)**

This function aborts the receive data which was using IRQ.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the <i>flexio_uart_handle_t</i> structure to store the transfer state.

32.8.7.25 `status_t FLEXIO_UART_TransferGetReceiveCount (FLEXIO_UART_Type *
base, flexio_uart_handle_t * handle, size_t * count)`

This function gets the number of bytes received driven by interrupt.

Parameters

<i>base</i>	Pointer to the FLEXIO_UART_Type structure.
<i>handle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.
<i>count</i>	Number of bytes received so far by the non-blocking transaction.

Return values

<i>kStatus_NoTransferInProgress</i>	transfer has finished or no transfer in progress.
<i>kStatus_Success</i>	Successfully return the count.

32.8.7.26 void FLEXIO_UART_TransferHandleIRQ (void * *uartType*, void * *uartHandle*)

This function processes the FlexIO UART transmit and receives the IRQ request.

Parameters

<i>uartType</i>	Pointer to the FLEXIO_UART_Type structure.
<i>uartHandle</i>	Pointer to the flexio_uart_handle_t structure to store the transfer state.

32.8.8 FlexIO eDMA UART Driver

32.8.8.1 Overview

Data Structures

- struct `flexio_uart_edma_handle_t`
UART eDMA handle. *More...*

Typedefs

- typedef void(* `flexio_uart_edma_transfer_callback_t`)(`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle, `status_t` status, void *userData)
UART transfer callback function.

Driver version

- #define `FSL_FLEXIO_UART_EDMA_DRIVER_VERSION` (`MAKE_VERSION`(2, 4, 1))
FlexIO UART EDMA driver version.

eDMA transactional

- `status_t FLEXIO_UART_TransferCreateHandleEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle, `flexio_uart_edma_transfer_callback_t` callback, void *userData, `edma_handle_t` *txEdmaHandle, `edma_handle_t` *rxEdmaHandle)
Initializes the UART handle which is used in transactional functions.
- `status_t FLEXIO_UART_TransferSendEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle, `flexio_uart_transfer_t` *xfer)
Sends data using eDMA.
- `status_t FLEXIO_UART_TransferReceiveEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle, `flexio_uart_transfer_t` *xfer)
Receives data using eDMA.
- void `FLEXIO_UART_TransferAbortSendEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle)
Aborts the sent data which using eDMA.
- void `FLEXIO_UART_TransferAbortReceiveEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle)
Aborts the receive data which using eDMA.
- `status_t FLEXIO_UART_TransferGetSendCountEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle, `size_t` *count)
Gets the number of bytes sent out.
- `status_t FLEXIO_UART_TransferGetReceiveCountEDMA` (`FLEXIO_UART_Type` *base, `flexio_uart_edma_handle_t` *handle, `size_t` *count)
Gets the number of bytes received.

32.8.8.2 Data Structure Documentation

32.8.8.2.1 struct_flexio_uart_edma_handle

Data Fields

- [flexio_uart_edma_transfer_callback_t](#) *callback*
Callback function.
- void * [userData](#)
UART callback function parameter.
- size_t [txDataSizeAll](#)
Total bytes to be sent.
- size_t [rxDataSizeAll](#)
Total bytes to be received.
- [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) [flexio_uart_edma_transfer_callback_t](#) [flexio_uart_edma_handle_t::callback](#)
- (2) void* [flexio_uart_edma_handle_t::userData](#)
- (3) size_t [flexio_uart_edma_handle_t::txDataSizeAll](#)
- (4) size_t [flexio_uart_edma_handle_t::rxDataSizeAll](#)
- (5) [edma_handle_t](#)* [flexio_uart_edma_handle_t::txEdmaHandle](#)
- (6) [edma_handle_t](#)* [flexio_uart_edma_handle_t::rxEdmaHandle](#)
- (7) uint8_t [flexio_uart_edma_handle_t::nbytes](#)
- (8) volatile uint8_t [flexio_uart_edma_handle_t::txState](#)

32.8.8.3 Macro Definition Documentation

32.8.8.3.1 #define FSL_FLEXIO_UART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 4, 1))

32.8.8.4 Typedef Documentation

32.8.8.4.1 `typedef void(* flexio_uart_edma_transfer_callback_t)(FLEXIO_UART_Type *base, flexio_uart_edma_handle_t *handle, status_t status, void *userData)`

32.8.8.5 Function Documentation

32.8.8.5.1 `status_t FLEXIO_UART_TransferCreateHandleEDMA (FLEXIO_UART_Type * base, flexio_uart_edma_handle_t * handle, flexio_uart_edma_transfer_callback_t callback, void * userData, edma_handle_t * txEdmaHandle, edma_handle_t * rxEdmaHandle)`

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type .
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure.
<i>callback</i>	The callback function.
<i>userData</i>	The parameter of the callback function.
<i>rxEdmaHandle</i>	User requested DMA handle for RX DMA transfer.
<i>txEdmaHandle</i>	User requested DMA handle for TX DMA transfer.

Return values

<i>kStatus_Success</i>	Successfully create the handle.
<i>kStatus_OutOfRange</i>	The FlexIO SPI eDMA type/handle table out of range.

32.8.8.5.2 `status_t FLEXIO_UART_TransferSendEDMA (FLEXIO_UART_Type * base, flexio_uart_edma_handle_t * handle, flexio_uart_transfer_t * xfer)`

This function sends data using eDMA. This is a non-blocking function, which returns right away. When all data is sent out, the send callback function is called.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	UART handle pointer.
<i>xfer</i>	UART eDMA transfer structure, see flexio_uart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_FLEXIO_UART-TxBusy</i>	Previous transfer on going.

32.8.8.5.3 **status_t FLEXIO_UART_TransferReceiveEDMA (FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, flexio_uart_transfer_t * *xfer*)**

This function receives data using eDMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>xfer</i>	UART eDMA transfer structure, see flexio_uart_transfer_t .

Return values

<i>kStatus_Success</i>	if succeed, others failed.
<i>kStatus_UART_RxBusy</i>	Previous transfer on going.

32.8.8.5.4 **void FLEXIO_UART_TransferAbortSendEDMA (FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*)**

This function aborts sent data which using eDMA.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure

32.8.8.5.5 **void FLEXIO_UART_TransferAbortReceiveEDMA (FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*)**

This function aborts the receive data which using eDMA.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure

32.8.8.5.6 status_t FLEXIO_UART_TransferGetSendCountEDMA (FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, size_t * *count*)

This function gets the number of bytes sent out.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>count</i>	Number of bytes sent so far by the non-blocking transaction.

Return values

<i>kStatus_NoTransferInProgress</i>	transfer has finished or no transfer in progress.
<i>kStatus_Success</i>	Successfully return the count.

32.8.8.5.7 status_t FLEXIO_UART_TransferGetReceiveCountEDMA (FLEXIO_UART_Type * *base*, flexio_uart_edma_handle_t * *handle*, size_t * *count*)

This function gets the number of bytes received.

Parameters

<i>base</i>	Pointer to FLEXIO_UART_Type
<i>handle</i>	Pointer to flexio_uart_edma_handle_t structure
<i>count</i>	Number of bytes received so far by the non-blocking transaction.

Return values

<i>kStatus_NoTransferInProgress</i>	transfer has finished or no transfer in progress.
-------------------------------------	---

<i>kStatus_Success</i>	Successfully return the count.
------------------------	--------------------------------

Chapter 33

FLEXRAM: on-chip RAM manager

33.1 Overview

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

The FLEXRAM module integrates the ITCM, DTCM, and OCRAM controllers, and supports parameterized RAM array and RAM array portioning.

This example code shows how to allocate RAM using the FLEXRAM driver.

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

Data Structures

- struct [flexram_ocram_ecc_single_error_info_t](#)
FLEXRAM ocram ecc single error information, including single error information, error address, error data. [More...](#)
- struct [flexram_ocram_ecc_multi_error_info_t](#)
FLEXRAM ocram ecc multiple error information, including multiple error information, error address, error data. [More...](#)
- struct [flexram_itcm_ecc_single_error_info_t](#)
FLEXRAM itcm ecc single error information, including single error information, error address, error data. [More...](#)
- struct [flexram_itcm_ecc_multi_error_info_t](#)
FLEXRAM itcm ecc multiple error information, including multiple error information, error address, error data. [More...](#)
- struct [flexram_dtcn_ecc_single_error_info_t](#)
FLEXRAM dtcm ecc single error information, including single error information, error address, error data. [More...](#)
- struct [flexram_dtcn_ecc_multi_error_info_t](#)
FLEXRAM dtcm ecc multiple error information, including multiple error information, error address, error data. [More...](#)

Macros

- #define [FLEXRAM_ECC_ERROR_DETAILED_INFO](#) 0U /* Define to zero means get raw ECC error information, which needs parse it by user. */
Get ECC error detailed information.

Enumerations

- enum {
 [kFLEXRAM_Read](#) = 0U,
 [kFLEXRAM_Write](#) = 1U }
Flexram write/read selection.

- enum {
[kFLEXRAM_OCRAMAccessError](#) = FLEXRAM_INT_STATUS_OCRAM_ERR_STATUS_MASK,
[kFLEXRAM_DTCMAccessError](#) = FLEXRAM_INT_STATUS_DTCM_ERR_STATUS_MASK,
[kFLEXRAM_ITCMAccessError](#) = FLEXRAM_INT_STATUS_ITCM_ERR_STATUS_MASK,
[kFLEXRAM_OCRAMMagicAddrMatch](#) = FLEXRAM_INT_STATUS_OCRAM_MAM_STATUS_MASK,
[kFLEXRAM_DTCMMagicAddrMatch](#) = FLEXRAM_INT_STATUS_DTCM_MAM_STATUS_MASK,
[kFLEXRAM_ITCMMagicAddrMatch](#) = FLEXRAM_INT_STATUS_ITCM_MAM_STATUS_MASK }
Interrupt status flag mask.
- enum [flexram_tcm_access_mode_t](#) {
[kFLEXRAM_TCMAccessFastMode](#) = 0U,
[kFLEXRAM_TCMAccessWaitMode](#) = 1U }
FLEXRAM TCM access mode.
- enum {
[kFLEXRAM_TCMSize32KB](#) = 32 * 1024U,
[kFLEXRAM_TCMSize64KB](#) = 64 * 1024U,
[kFLEXRAM_TCMSize128KB](#) = 128 * 1024U,
[kFLEXRAM_TCMSize256KB](#) = 256 * 1024U,
[kFLEXRAM_TCMSize512KB](#) = 512 * 1024U }
FLEXRAM TCM support size.

Functions

- static void [FLEXRAM_SetTCMReadAccessMode](#) (FLEXRAM_Type *base, [flexram_tcm_access_mode_t](#) mode)
FLEXRAM module sets TCM read access mode.
- static void [FLEXRAM_SetTCMWriteAccessMode](#) (FLEXRAM_Type *base, [flexram_tcm_access_mode_t](#) mode)
FLEXRAM module set TCM write access mode.
- static void [FLEXRAM_EnableForceRamClockOn](#) (FLEXRAM_Type *base, bool enable)
FLEXRAM module force ram clock on.
- static void [FLEXRAM_SetOCRAMMagicAddr](#) (FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)
FLEXRAM OCRAM magic addr configuration.
- static void [FLEXRAM_SetDTCMMagicAddr](#) (FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)
FLEXRAM DTCM magic addr configuration.
- static void [FLEXRAM_SetITCMMagicAddr](#) (FLEXRAM_Type *base, uint16_t magicAddr, uint32_t rwSel)
FLEXRAM ITCM magic addr configuration.
- void [FLEXRAM_EnableECC](#) (FLEXRAM_Type *base, bool OcramECCEnable, bool TcmECCEnable)
FLEXRAM get ocram ecc single error information.
- void [FLEXRAM_GetOcramSingleErrorInfo](#) (FLEXRAM_Type *base, [flexram_ocram_ecc_single_error_info_t](#) *info)

- FLEXRAM get ocram ecc single error information.*
- void [FLEXRAM_GetOcramMultiErrInfo](#) (FLEXRAM_Type *base, [flexram_ocram_ecc_multi_error_info_t](#) *info)
- FLEXRAM get ocram ecc multiple error information.*
- void [FLEXRAM_GetItcmSingleErrInfo](#) (FLEXRAM_Type *base, [flexram_itcm_ecc_single_error_info_t](#) *info)
- FLEXRAM get itcm ecc single error information.*
- void [FLEXRAM_GetItcmMultiErrInfo](#) (FLEXRAM_Type *base, [flexram_itcm_ecc_multi_error_info_t](#) *info)
- FLEXRAM get itcm ecc multiple error information.*
- void [FLEXRAM_GetDtcSingleErrInfo](#) (FLEXRAM_Type *base, [flexram_dtc_ecc_single_error_info_t](#) *info, uint8_t bank)
- FLEXRAM get d0tcm ecc single error information.*
- void [FLEXRAM_GetDtcMultiErrInfo](#) (FLEXRAM_Type *base, [flexram_dtc_ecc_multi_error_info_t](#) *info, uint8_t bank)
- FLEXRAM get d0tcm ecc multiple error information.*

Driver version

- #define [FSL_FLEXRAM_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2U, 1U, 0U))
Driver version 2.1.0.

Initialization and de-initialization

- void [FLEXRAM_Init](#) (FLEXRAM_Type *base)
FLEXRAM module initialization function.
- void [FLEXRAM_Deinit](#) (FLEXRAM_Type *base)
De-initializes the FLEXRAM.

Status

- static uint32_t [FLEXRAM_GetInterruptStatus](#) (FLEXRAM_Type *base)
FLEXRAM module gets interrupt status.
- static void [FLEXRAM_ClearInterruptStatus](#) (FLEXRAM_Type *base, uint32_t status)
FLEXRAM module clears interrupt status.
- static void [FLEXRAM_EnableInterruptStatus](#) (FLEXRAM_Type *base, uint32_t status)
FLEXRAM module enables interrupt status.
- static void [FLEXRAM_DisableInterruptStatus](#) (FLEXRAM_Type *base, uint32_t status)
FLEXRAM module disable interrupt status.

Interrupts

- static void [FLEXRAM_EnableInterruptSignal](#) (FLEXRAM_Type *base, uint32_t status)
FLEXRAM module enables interrupt.
- static void [FLEXRAM_DisableInterruptSignal](#) (FLEXRAM_Type *base, uint32_t status)
FLEXRAM module disables interrupt.

33.2 Data Structure Documentation

33.2.1 struct flexram_ocram_ecc_single_error_info_t

Data Fields

- uint32_t [OcramSingleErrorInfo](#)
Ocram single error information, user should parse it by themselves.
- uint32_t [OcramSingleErrorAddr](#)
Ocram single error address.
- uint32_t [OcramSingleErrorDataLSB](#)
Ocram single error data LSB.
- uint32_t [OcramSingleErrorDataMSB](#)
Ocram single error data MSB.

Field Documentation

(1) uint32_t flexram_ocram_ecc_single_error_info_t::OcramSingleErrorInfo

33.2.2 struct flexram_ocram_ecc_multi_error_info_t

Data Fields

- uint32_t [OcramMultiErrorInfo](#)
Ocram single error information, user should parse it by themselves.
- uint32_t [OcramMultiErrorAddr](#)
Ocram multiple error address.
- uint32_t [OcramMultiErrorDataLSB](#)
Ocram multiple error data LSB.
- uint32_t [OcramMultiErrorDataMSB](#)
Ocram multiple error data MSB.

Field Documentation

(1) uint32_t flexram_ocram_ecc_multi_error_info_t::OcramMultiErrorInfo

33.2.3 struct flexram_itcm_ecc_single_error_info_t

Data Fields

- uint32_t [ItcmSingleErrorInfo](#)
itcm single error information, user should parse it by themselves.
- uint32_t [ItcmSingleErrorAddr](#)
itcm single error address
- uint32_t [ItcmSingleErrorDataLSB](#)
itcm single error data LSB
- uint32_t [ItcmSingleErrorDataMSB](#)
itcm single error data MSB

Field Documentation

(1) `uint32_t flexram_itcm_ecc_single_error_info_t::ItcmSingleErrorInfo`

33.2.4 struct flexram_itcm_ecc_multi_error_info_t

Data Fields

- `uint32_t ItcmMultiErrorInfo`
itcm multiple error information, user should parse it by themselves.
- `uint32_t ItcmMultiErrorAddr`
itcm multiple error address
- `uint32_t ItcmMultiErrorDataLSB`
itcm multiple error data LSB
- `uint32_t ItcmMultiErrorDataMSB`
itcm multiple error data MSB

Field Documentation

(1) `uint32_t flexram_itcm_ecc_multi_error_info_t::ItcmMultiErrorInfo`

33.2.5 struct flexram_dtcn_ecc_single_error_info_t

Data Fields

- `uint32_t DtcnSingleErrorInfo`
dtcm single error information, user should parse it by themselves.
- `uint32_t DtcnSingleErrorAddr`
dtcm single error address
- `uint32_t DtcnSingleErrorData`
dtcm single error data

Field Documentation

(1) `uint32_t flexram_dtcn_ecc_single_error_info_t::DtcnSingleErrorInfo`

33.2.6 struct flexram_dtcn_ecc_multi_error_info_t

Data Fields

- `uint32_t DtcnMultiErrorInfo`
dtcm multiple error information, user should parse it by themselves.
- `uint32_t DtcnMultiErrorAddr`
dtcm multiple error address
- `uint32_t DtcnMultiErrorData`
dtcm multiple error data

Field Documentation

(1) `uint32_t flexram_dtcmm_ecc_multi_error_info_t::DtcmmMultiErrorInfo`

33.3 Macro Definition Documentation

33.3.1 `#define FSL_FLEXRAM_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 0U))`

33.3.2 `#define FLEXRAM_ECC_ERROR_DETAILED_INFO 0U /* Define to zero means get raw ECC error information, which needs parse it by user. */`

33.4 Enumeration Type Documentation

33.4.1 anonymous enum

Enumerator

kFLEXRAM_Read read
kFLEXRAM_Write write

33.4.2 anonymous enum

Enumerator

kFLEXRAM_OCRAMAccessError OCRAM accesses unallocated address.
kFLEXRAM_DTCMAccessError DTCM accesses unallocated address.
kFLEXRAM_ITCMAccessError ITCM accesses unallocated address.
kFLEXRAM_OCRAMMagicAddrMatch OCRAM magic address match.
kFLEXRAM_DTCMMagicAddrMatch DTCM magic address match.
kFLEXRAM_ITCMMagicAddrMatch ITCM magic address match.

33.4.3 enum flexram_tcm_access_mode_t

Fast access mode expected to be finished in 1-cycle; Wait access mode expected to be finished in 2-cycle. Wait access mode is a feature of the flexram and it should be used when the CPU clock is too fast to finish TCM access in 1-cycle. Normally, fast mode is the default mode, the efficiency of the TCM access will be better.

Enumerator

kFLEXRAM_TCMAccessFastMode fast access mode
kFLEXRAM_TCMAccessWaitMode wait access mode

33.4.4 anonymous enum

Enumerator

kFLEXRAM_TCMSize32KB TCM total size be 32KB.
kFLEXRAM_TCMSize64KB TCM total size be 64KB.
kFLEXRAM_TCMSize128KB TCM total size be 128KB.
kFLEXRAM_TCMSize256KB TCM total size be 256KB.
kFLEXRAM_TCMSize512KB TCM total size be 512KB.

33.5 Function Documentation

33.5.1 void FLEXRAM_Init (FLEXRAM_Type * *base*)

Parameters

<i>base</i>	FLEXRAM base address.
-------------	-----------------------

33.5.2 static uint32_t FLEXRAM_GetInterruptStatus (FLEXRAM_Type * *base*) [inline], [static]

Parameters

<i>base</i>	FLEXRAM base address.
-------------	-----------------------

33.5.3 static void FLEXRAM_ClearInterruptStatus (FLEXRAM_Type * *base*, uint32_t *status*) [inline], [static]

Parameters

<i>base</i>	FLEXRAM base address.
<i>status</i>	Status to be cleared.

33.5.4 static void FLEXRAM_EnableInterruptStatus (FLEXRAM_Type * *base*, uint32_t *status*) [inline], [static]

Parameters

<i>base</i>	FLEXRAM base address.
<i>status</i>	Status to be enabled.

**33.5.5 static void FLEXRAM_DisableInterruptStatus (FLEXRAM_Type * *base*,
uint32_t *status*) [inline], [static]**

Parameters

<i>base</i>	FLEXRAM base address.
<i>status</i>	Status to be disabled.

**33.5.6 static void FLEXRAM_EnableInterruptSignal (FLEXRAM_Type * *base*,
uint32_t *status*) [inline], [static]**

Parameters

<i>base</i>	FLEXRAM base address.
<i>status</i>	Status interrupt to be enabled.

**33.5.7 static void FLEXRAM_DisableInterruptSignal (FLEXRAM_Type * *base*,
uint32_t *status*) [inline], [static]**

Parameters

<i>base</i>	FLEXRAM base address.
<i>status</i>	Status interrupt to be disabled.

**33.5.8 static void FLEXRAM_SetTCMReadAccessMode (FLEXRAM_Type * *base*,
flexram_tcm_access_mode_t *mode*) [inline], [static]**

Parameters

<i>base</i>	FLEXRAM base address.
<i>mode</i>	Access mode.

33.5.9 static void FLEXRAM_SetTCMWriteAccessMode (FLEXRAM_Type * *base*, flexram_tcm_access_mode_t *mode*) [inline], [static]

Parameters

<i>base</i>	FLEXRAM base address.
<i>mode</i>	Access mode.

33.5.10 static void FLEXRAM_EnableForceRamClockOn (FLEXRAM_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	FLEXRAM base address.
<i>enable</i>	Enable or disable clock force on.

33.5.11 static void FLEXRAM_SetOCRAMMagicAddr (FLEXRAM_Type * *base*, uint16_t *magicAddr*, uint32_t *rwSel*) [inline], [static]

When read/write access hit magic address, it will generate interrupt.

Parameters

<i>base</i>	FLEXRAM base address.
<i>magicAddr</i>	Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].

<i>rwSel</i>	Read/write selection. 0 for read access while 1 for write access.
--------------	---

33.5.12 static void FLEXRAM_SetDTCMMagicAddr (FLEXRAM_Type * *base*, uint16_t *magicAddr*, uint32_t *rwSel*) [inline], [static]

When read/write access hits magic address, it will generate interrupt.

Parameters

<i>base</i>	FLEXRAM base address.
<i>magicAddr</i>	Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
<i>rwSel</i>	Read/write selection. 0 for read access while 1 write access.

33.5.13 static void FLEXRAM_SetITCMMagicAddr (FLEXRAM_Type * *base*, uint16_t *magicAddr*, uint32_t *rwSel*) [inline], [static]

When read/write access hits magic address, it will generate interrupt.

Parameters

<i>base</i>	FLEXRAM base address.
<i>magicAddr</i>	Magic address, the actual address bits [18:3] is corresponding to the register field [16:1].
<i>rwSel</i>	Read/write selection. 0 for read access while 1 for write access.

33.5.14 void FLEXRAM_EnableECC (FLEXRAM_Type * *base*, bool *OcramECCEnable*, bool *TcmECCEnable*)

Parameters

<i>base</i>	FLEXRAM base address.
-------------	-----------------------

<i>OcramECC-Enable</i>	ocram ecc enablement.
<i>TcmECC-Enable</i>	tcm(itcm/d0tcm/d1tcm) ecc enablement.

33.5.15 void FLEXRAM_GetOcramSingleErroInfo (FLEXRAM_Type * *base*, flexram_ocram_ecc_single_error_info_t * *info*)

Parameters

<i>base</i>	FLEXRAM base address.
<i>info</i>	ecc error information.

33.5.16 void FLEXRAM_GetOcramMultiErroInfo (FLEXRAM_Type * *base*, flexram_ocram_ecc_multi_error_info_t * *info*)

Parameters

<i>base</i>	FLEXRAM base address.
<i>info</i>	ecc error information.

33.5.17 void FLEXRAM_GetItcmSingleErroInfo (FLEXRAM_Type * *base*, flexram_itcm_ecc_single_error_info_t * *info*)

Parameters

<i>base</i>	FLEXRAM base address.
<i>info</i>	ecc error information.

33.5.18 void FLEXRAM_GetItcmMultiErroInfo (FLEXRAM_Type * *base*, flexram_itcm_ecc_multi_error_info_t * *info*)

Parameters

<i>base</i>	FLEXRAM base address.
<i>info</i>	ecc error information.

33.5.19 void FLEXRAM_GetDtcSingleErrorInfo (FLEXRAM_Type * *base*, flexram_dtc_ecc_single_error_info_t * *info*, uint8_t *bank*)

Parameters

<i>base</i>	FLEXRAM base address.
<i>info</i>	ecc error information.
<i>bank</i>	DTCM bank, 0 is D0TCM, 1 is D1TCM.

33.5.20 void FLEXRAM_GetDtcMultiErrorInfo (FLEXRAM_Type * *base*, flexram_dtc_ecc_multi_error_info_t * *info*, uint8_t *bank*)

Parameters

<i>base</i>	FLEXRAM base address.
<i>info</i>	ecc error information.
<i>bank</i>	DTCM bank, 0 is D0TCM, 1 is D1TCM.

Chapter 34

FLEXSPI: Flexible Serial Peripheral Interface Driver

34.1 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 = 0xFFFU }

```

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

Trigger source of current command sequence granted by arbitrator.
- enum flexspi_command_type_t {


```

kFLEXSPI_Command,
kFLEXSPI_Config }

```

Command type.

Driver version

- #define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 3, 5))

FLEXSPI driver version 2.3.5.

Initialization and deinitialization

- uint32_t FLEXSPI_GetInstance (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, uint32_t *buffer, size_t size)
Sends a buffer of data bytes using blocking method.
- [status_t](#) [FLEXSPI_ReadBlocking](#) (FLEXSPI_Type *base, uint32_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.

34.2 Data Structure Documentation

34.2.1 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 ahbGrantTimeoutCyle*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 ahbGrantTimeoutCyle*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::enableAHBWritelpTxFifo
- (13) bool flexspi_config_t::enableAHBWritelpRxFifo
- (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`

34.2.2 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`

34.2.3 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`

34.2.4 struct _flexspi_handle

Data Fields

- `uint32_t state`
Internal state for FLEXSPI transfer.
- `uint32_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) `uint32_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`

34.3 Macro Definition Documentation

34.3.1 #define FSL_FLEXSPI_DRIVER_VERSION (MAKE_VERSION(2, 3, 5))

34.3.2 #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))
```

34.4 Typedef Documentation

34.4.1 `typedef void(* flexspi_transfer_callback_t)(FLEXSPI_Type *base, flexspi_handle_t *handle, status_t status, void *userData)`

34.5 Enumeration Type Documentation

34.5.1 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.

34.5.2 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.

34.5.3 enum flexspi_pad_t

Enumerator

kFLEXSPI_1PAD Transmit command/address and transmit/receive data only through DATA0/D-ATA1.
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].

34.5.4 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.

34.5.5 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.

34.5.6 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.

34.5.7 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.

34.5.8 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.

34.5.9 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.

34.5.10 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.

34.5.11 enum flexspi_arb_command_source_t

34.5.12 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.

34.6 Function Documentation

34.6.1 uint32_t FLEXSPI_GetInstance (FLEXSPI_Type * *base*)

Parameters

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

34.6.2 status_t FLEXSPI_CheckAndClearError (FLEXSPI_Type * *base*, uint32_t *status*)

Parameters

<i>base</i>	FLEXSPI base pointer.
<i>status</i>	interrupt status.

34.6.3 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.

34.6.4 void FLEXSPI_GetDefaultConfig (flexspi_config_t * *config*)

Parameters

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

34.6.5 void FLEXSPI_Deinit (FLEXSPI_Type * *base*)

Clears the FLEXSPI state and FLEXSPI module registers.

Parameters

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

34.6.6 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.

34.6.7 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.

34.6.8 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.
-------------	----------------------------------

34.6.9 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.

34.6.10 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.

34.6.11 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.

34.6.12 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.

34.6.13 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.

34.6.14 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.
------------	------------------

34.6.15 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.
------------	------------------

34.6.16 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.

34.6.17 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.

34.6.18 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 flexspi_flags_t could get the related status.
------------------	---

34.6.19 `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.

34.6.20 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.
----------------	-------------------------------------

34.6.21 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.
--------------	--------------------------------------

34.6.22 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.
--------------	---------------------------------------

**34.6.23 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.

**34.6.24 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 flexspi_read_sample_clock_t

**34.6.25 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.

**34.6.26 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.

34.6.27 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.

34.6.28 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.

34.6.29 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.

34.6.30 **status_t FLEXSPI_WriteBlocking (FLEXSPI_Type * *base*, uint32_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

34.6.31 **status_t FLEXSPI_ReadBlocking (FLEXSPI_Type * *base*, uint32_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_SequenceExecution-Timeout</i>	sequence execution timeout
<i>kStatus_FLEXSPI_Ip-CommandSequenceError</i>	IP command sequencen error detected
<i>kStatus_FLEXSPI_Ip-CommandGrantTimeout</i>	IP command grant timeout detected

34.6.32 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_SequenceExecution-Timeout</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

34.6.33 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.

34.6.34 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 flexspi_handle_t structure which stores the transfer state.
<i>xfer</i>	pointer to flexspi_transfer_t structure.

Return values

<i>kStatus_Success</i>	Successfully start the data transmission.
<i>kStatus_FLEXSPI_Busy</i>	Previous transmission still not finished.

34.6.35 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 flexspi_handle_t structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the non-blocking transaction.

Return values

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

34.6.36 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 flexspi_handle_t structure which stores the transfer state

34.6.37 void FLEXSPI_TransferHandleIRQ (FLEXSPI_Type * *base*, flexspi_handle_t * *handle*)

Parameters

<i>base</i>	FLEXSPI peripheral base address.
<i>handle</i>	pointer to flexspi_handle_t structure.

Chapter 35

GPT: General Purpose Timer

35.1 Overview

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

35.2 Function groups

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

35.2.1 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.

35.3 Typical use case

35.3.1 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, 3)`)

Initialization and deinitialization

- void `GPT_Init` (GPT_Type *base, const `gpt_config_t` *initConfig)
Initialize GPT to reset state and initialize running mode.

- void [GPT_Deinit](#) (GPT_Type *base)
Disables the module and gates the GPT clock.
- void [GPT_GetDefaultConfig](#) (gpt_config_t *config)
Fills in the GPT configuration structure with default settings.

Software Reset

- static void [GPT_SoftwareReset](#) (GPT_Type *base)
Software reset of GPT module.

Clock source and frequency control

- static void [GPT_SetClockSource](#) (GPT_Type *base, gpt_clock_source_t gptClkSource)
Set clock source of GPT.
- static gpt_clock_source_t [GPT_GetClockSource](#) (GPT_Type *base)
Get clock source of GPT.
- static void [GPT_SetClockDivider](#) (GPT_Type *base, uint32_t divider)
Set pre scaler of GPT.
- static uint32_t [GPT_GetClockDivider](#) (GPT_Type *base)
Get clock divider in GPT module.
- static void [GPT_SetOscClockDivider](#) (GPT_Type *base, uint32_t divider)
OSC 24M pre-scaler before selected by clock source.
- static uint32_t [GPT_GetOscClockDivider](#) (GPT_Type *base)
Get OSC 24M clock divider in GPT module.

Timer Start and Stop

- static void [GPT_StartTimer](#) (GPT_Type *base)
Start GPT timer.
- static void [GPT_StopTimer](#) (GPT_Type *base)
Stop GPT timer.

Read the timer period

- static uint32_t [GPT_GetCurrentTimerCount](#) (GPT_Type *base)
Reads the current GPT counting value.

GPT Input/Output Signal Control

- static void [GPT_SetInputOperationMode](#) (GPT_Type *base, gpt_input_capture_channel_t channel, gpt_input_operation_mode_t mode)
Set GPT operation mode of input capture channel.
- static gpt_input_operation_mode_t [GPT_GetInputOperationMode](#) (GPT_Type *base, gpt_input_capture_channel_t channel)
Get GPT operation mode of input capture channel.
- static uint32_t [GPT_GetInputCaptureValue](#) (GPT_Type *base, gpt_input_capture_channel_t channel)
Get GPT input capture value of certain channel.
- static void [GPT_SetOutputOperationMode](#) (GPT_Type *base, gpt_output_compare_channel_t channel, gpt_output_operation_mode_t mode)

- *Set GPT operation mode of output compare channel.*
static [gpt_output_operation_mode_t](#) [GPT_GetOutputOperationMode](#) (GPT_Type *base, [gpt_output_compare_channel_t](#) channel)
- *Get GPT operation mode of output compare channel.*
static void [GPT_SetOutputCompareValue](#) (GPT_Type *base, [gpt_output_compare_channel_t](#) channel, uint32_t value)
- *Set GPT output compare value of output compare channel.*
static uint32_t [GPT_GetOutputCompareValue](#) (GPT_Type *base, [gpt_output_compare_channel_t](#) channel)
- *Get GPT output compare value of output compare channel.*
static void [GPT_ForceOutput](#) (GPT_Type *base, [gpt_output_compare_channel_t](#) channel)
Force GPT output action on output compare channel, ignoring comparator.

GPT Interrupt and Status Interface

- static void [GPT_EnableInterrupts](#) (GPT_Type *base, uint32_t mask)
Enables the selected GPT interrupts.
- static void [GPT_DisableInterrupts](#) (GPT_Type *base, uint32_t mask)
Disables the selected GPT interrupts.
- static uint32_t [GPT_GetEnabledInterrupts](#) (GPT_Type *base)
Gets the enabled GPT interrupts.

Status Interface

- static uint32_t [GPT_GetStatusFlags](#) (GPT_Type *base, [gpt_status_flag_t](#) flags)
Get GPT status flags.
- static void [GPT_ClearStatusFlags](#) (GPT_Type *base, [gpt_status_flag_t](#) flags)
Clears the GPT status flags.

35.4 Data Structure Documentation

35.4.1 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`

35.5 Enumeration Type Documentation

35.5.1 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.

35.5.2 enum `gpt_input_capture_channel_t`

Enumerator

kGPT_InputCapture_Channel1 GPT Input Capture Channel1.
kGPT_InputCapture_Channel2 GPT Input Capture Channel2.

35.5.3 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.

35.5.4 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.

35.5.5 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.

35.5.6 enum gpt_interrupt_enable_t

Enumerator

kGPT_OutputCompare1InterruptEnable Output Compare Channel1 interrupt enable.
kGPT_OutputCompare2InterruptEnable Output Compare Channel2 interrupt enable.
kGPT_OutputCompare3InterruptEnable Output Compare Channel3 interrupt enable.
kGPT_InputCapture1InterruptEnable Input Capture Channel1 interrupt enable.
kGPT_InputCapture2InterruptEnable Input Capture Channel1 interrupt enable.
kGPT_RollOverFlagInterruptEnable Counter rolled over interrupt enable.

35.5.7 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.

35.6 Function Documentation

35.6.1 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.

35.6.2 void GPT_Deinit (GPT_Type * *base*)

Parameters

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

35.6.3 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.
---------------	--

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

Parameters

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

35.6.5 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 gpt_clock_source_t typedef enumeration).

35.6.6 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).

35.6.7 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).

35.6.8 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).

35.6.9 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).

35.6.10 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).

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

Parameters

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

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

Parameters

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

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

Parameters

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

Returns

Current GPT counter value.

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

35.6.15 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 gpt_input_capture_channel_t typedef enumeration).

Returns

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

35.6.16 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 gpt_input_capture_channel_t typedef enumeration).

Returns

GPT input capture value.

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

35.6.18 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 gpt_output_compare_channel_t typedef enumeration).

Returns

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

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

**35.6.20 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 gpt_output_compare_channel_t typedef enumeration).

Returns

GPT output compare value.

**35.6.21 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 gpt_output_compare_channel_t typedef enumeration).

35.6.22 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 gpt_interrupt_enable_t

35.6.23 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 gpt_interrupt_enable_t

35.6.24 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](#)

35.6.25 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 gpt_status_flag_t for bit definition).

Returns

GPT status, each bit represents one status flag.

35.6.26 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 gpt_status_flag_t for bit definition).

Chapter 36

GPIO: General-Purpose Input/Output Driver

36.1 Overview

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

36.2 Typical use case

36.2.1 Input Operation

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

Data Structures

- struct `gpio_pin_config_t`
GPIO Init structure definition. [More...](#)

Enumerations

- enum `gpio_pin_direction_t` {
 `kGPIO_DigitalInput` = 0U,
 `kGPIO_DigitalOutput` = 1U }
GPIO direction definition.
- enum `gpio_interrupt_mode_t` {
 `kGPIO_NoIntmode` = 0U,
 `kGPIO_IntLowLevel` = 1U,
 `kGPIO_IntHighLevel` = 2U,
 `kGPIO_IntRisingEdge` = 3U,
 `kGPIO_IntFallingEdge` = 4U,
 `kGPIO_IntRisingOrFallingEdge` = 5U }
GPIO interrupt mode definition.

Driver version

- #define `FSL_GPIO_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)
GPIO driver version.

GPIO Initialization and Configuration functions

- void `GPIO_PinInit` (`GPIO_Type *base`, `uint32_t pin`, const `gpio_pin_config_t *Config`)
Initializes the GPIO peripheral according to the specified parameters in the initConfig.

GPIO Reads and Write Functions

- void [GPIO_PinWrite](#) (GPIO_Type *base, uint32_t pin, uint8_t output)
Sets the output level of the individual GPIO pin to logic 1 or 0.
- static void [GPIO_WritePinOutput](#) (GPIO_Type *base, uint32_t pin, uint8_t output)
Sets the output level of the individual GPIO pin to logic 1 or 0.
- static void [GPIO_PortSet](#) (GPIO_Type *base, uint32_t mask)
Sets the output level of the multiple GPIO pins to the logic 1.
- static void [GPIO_SetPinsOutput](#) (GPIO_Type *base, uint32_t mask)
Sets the output level of the multiple GPIO pins to the logic 1.
- static void [GPIO_PortClear](#) (GPIO_Type *base, uint32_t mask)
Sets the output level of the multiple GPIO pins to the logic 0.
- static void [GPIO_ClearPinsOutput](#) (GPIO_Type *base, uint32_t mask)
Sets the output level of the multiple GPIO pins to the logic 0.
- static void [GPIO_PortToggle](#) (GPIO_Type *base, uint32_t mask)
Reverses the current output logic of the multiple GPIO pins.
- static uint32_t [GPIO_PinRead](#) (GPIO_Type *base, uint32_t pin)
Reads the current input value of the GPIO port.
- static uint32_t [GPIO_ReadPinInput](#) (GPIO_Type *base, uint32_t pin)
Reads the current input value of the GPIO port.

GPIO Reads Pad Status Functions

- static uint8_t [GPIO_PinReadPadStatus](#) (GPIO_Type *base, uint32_t pin)
Reads the current GPIO pin pad status.
- static uint8_t [GPIO_ReadPadStatus](#) (GPIO_Type *base, uint32_t pin)
Reads the current GPIO pin pad status.

Interrupts and flags management functions

- void [GPIO_PinSetInterruptConfig](#) (GPIO_Type *base, uint32_t pin, [gpio_interrupt_mode_t](#) pinInterruptMode)
Sets the current pin interrupt mode.
- static void [GPIO_SetPinInterruptConfig](#) (GPIO_Type *base, uint32_t pin, [gpio_interrupt_mode_t](#) pinInterruptMode)
Sets the current pin interrupt mode.
- static void [GPIO_PortEnableInterrupts](#) (GPIO_Type *base, uint32_t mask)
Enables the specific pin interrupt.
- static void [GPIO_EnableInterrupts](#) (GPIO_Type *base, uint32_t mask)
Enables the specific pin interrupt.
- static void [GPIO_PortDisableInterrupts](#) (GPIO_Type *base, uint32_t mask)
Disables the specific pin interrupt.
- static void [GPIO_DisableInterrupts](#) (GPIO_Type *base, uint32_t mask)
Disables the specific pin interrupt.
- static uint32_t [GPIO_PortGetInterruptFlags](#) (GPIO_Type *base)
Reads individual pin interrupt status.
- static uint32_t [GPIO_GetPinsInterruptFlags](#) (GPIO_Type *base)
Reads individual pin interrupt status.
- static void [GPIO_PortClearInterruptFlags](#) (GPIO_Type *base, uint32_t mask)
Clears pin interrupt flag.
- static void [GPIO_ClearPinsInterruptFlags](#) (GPIO_Type *base, uint32_t mask)

Clears pin interrupt flag.

36.3 Data Structure Documentation

36.3.1 struct gpio_pin_config_t

Data Fields

- [gpio_pin_direction_t direction](#)
Specifies the pin direction.
- uint8_t [outputLogic](#)
Set a default output logic, which has no use in input.
- [gpio_interrupt_mode_t interruptMode](#)
Specifies the pin interrupt mode, a value of [gpio_interrupt_mode_t](#).

Field Documentation

(1) [gpio_pin_direction_t gpio_pin_config_t::direction](#)

(2) [gpio_interrupt_mode_t gpio_pin_config_t::interruptMode](#)

36.4 Macro Definition Documentation

36.4.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

36.5 Enumeration Type Documentation

36.5.1 enum gpio_pin_direction_t

Enumerator

kGPIO_DigitalInput Set current pin as digital input.
kGPIO_DigitalOutput Set current pin as digital output.

36.5.2 enum gpio_interrupt_mode_t

Enumerator

kGPIO_NoIntmode Set current pin general IO functionality.
kGPIO_IntLowLevel Set current pin interrupt is low-level sensitive.
kGPIO_IntHighLevel Set current pin interrupt is high-level sensitive.
kGPIO_IntRisingEdge Set current pin interrupt is rising-edge sensitive.
kGPIO_IntFallingEdge Set current pin interrupt is falling-edge sensitive.
kGPIO_IntRisingOrFallingEdge Enable the edge select bit to override the ICR register's configuration.

36.6 Function Documentation

36.6.1 void GPIO_PinInit (GPIO_Type * *base*, uint32_t *pin*, const gpio_pin_config_t * *Config*)

Parameters

<i>base</i>	GPIO base pointer.
<i>pin</i>	Specifies the pin number
<i>Config</i>	pointer to a gpio_pin_config_t structure that contains the configuration information.

36.6.2 void GPIO_PinWrite (GPIO_Type * *base*, uint32_t *pin*, uint8_t *output*)

Parameters

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

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

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

36.6.4 static void GPIO_PortSet (GPIO_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

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

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

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

36.6.6 `static void GPIO_PortClear (GPIO_Type * base, uint32_t mask)`
`[inline], [static]`

Parameters

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

36.6.7 static void GPIO_ClearPinsOutput (GPIO_Type * *base*, uint32_t *mask*)
[inline], [static]

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

36.6.8 static void GPIO_PortToggle (GPIO_Type * *base*, uint32_t *mask*)
[inline], [static]

Parameters

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

36.6.9 static uint32_t GPIO_PinRead (GPIO_Type * *base*, uint32_t *pin*)
[inline], [static]

Parameters

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

Return values

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

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

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

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

Parameters

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

Return values

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

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

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

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

Parameters

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

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

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

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

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

Parameters

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

Return values

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

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

Parameters

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

Return values

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

36.6.21 **static void GPIO_PortClearInterruptFlags (GPIO_Type * *base*, uint32_t *mask*)** **[inline], [static]**

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

Parameters

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

36.6.22 **static void GPIO_ClearPinsInterruptFlags (GPIO_Type * *base*, uint32_t *mask*)** **[inline], [static]**

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

Parameters

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

Chapter 37

KPP: KeyPad Port Driver

37.1 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.

37.2 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.

37.3 Data Structure Documentation

37.3.1 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".

37.4 Macro Definition Documentation

37.4.1 #define FSL_KPP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

37.5 Enumeration Type Documentation

37.5.1 enum kpp_interrupt_enable_t

This enumeration uses one-hot 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.

37.5.2 enum kpp_sync_operation_t

Enumerator

kKPP_ClearKeyDepressSyncChain Keypad depress interrupt status.
kKPP_SetKeyReleasesSyncChain Keypad release interrupt status.

37.6 Function Documentation

37.6.1 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.

37.6.2 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.
-------------	------------------------------

37.6.3 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.

37.6.4 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.

37.6.5 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.

37.6.6 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.
-------------	--

37.6.7 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.

37.6.8 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 38

LCDIFv2: LCD Interface v2

38.1 Overview

The MCUXpresso SDK provides a peripheral driver for the LCD Interface v2(LCDIFv2)

The LCDIFv2 supports RGB mode (dot clock mode), it supports up to maximum 8 layers of alpha blending.

38.2 Shadow load

Shadow registers are used for LCDIFv2 layer configuration, when layer configurations are set, they are written to the shadow registers and do not take effect, after the function LCDIFV2_TriggerLayerShadowLoad is called, the new configuration are loaded to the active control registers at next vertical blank period. This mechanism ensures that all configurations are loaded at the same time.

38.3 Typical use case

Refer to the driver examples codes located at <SDK_ROOT>/boards/<BOARD>/driver_examples/lcdifv2

Data Structures

- struct [lcdifv2_display_config_t](#)
LCDIF v2 display configure structure. [More...](#)
- struct [lcdifv2_buffer_config_t](#)
LCDIF v2 source buffer configuration. [More...](#)
- struct [lcdifv2_blend_config_t](#)
LCDIF v2 layer alpha blending configuration. [More...](#)

Macros

- #define [LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT\(layer\)](#) (1U << ((uint32_t)(layer) + 24-U))
LCDIF v2 FIFO empty interrupt.
- #define [LCDIFV2_MAKE_DMA_DONE_INTERRUPT\(layer\)](#) (1U << ((uint32_t)(layer) + 16U))
LCDIF v2 DMA done interrupt.
- #define [LCDIFV2_MAKE_DMA_ERROR_INTERRUPT\(layer\)](#) (1U << ((uint32_t)(layer) + 8U))
LCDIF v2 DMA error interrupt.

Enumerations

- enum `_lcdifv2_polarity_flags` {
`kLCDIFV2_VsyncActiveLow` = `LCDIFV2_CTRL_INV_VS_MASK`,
`kLCDIFV2_VsyncActiveHigh` = `0U`,
`kLCDIFV2_HsyncActiveLow` = `LCDIFV2_CTRL_INV_HS_MASK`,
`kLCDIFV2_HsyncActiveHigh` = `0U`,
`kLCDIFV2_DataEnableActiveLow` = `LCDIFV2_CTRL_INV_DE_MASK`,
`kLCDIFV2_DataEnableActiveHigh` = `0U`,
`kLCDIFV2_DriveDataOnFallingClkEdge` = `LCDIFV2_CTRL_INV_PXCK_MASK`,
`kLCDIFV2_DriveDataOnRisingClkEdge` = `0U`,
`kLCDIFV2_DataActiveLow` = `LCDIFV2_CTRL_NEG_MASK`,
`kLCDIFV2_DataActiveHigh` = `0U` }
LCDIF v2 signal polarity flags.
- enum `_lcdifv2_interrupt` {
`kLCDIFV2_Layer0FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(0)`,
`kLCDIFV2_Layer1FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(1)`,
`kLCDIFV2_Layer2FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(2)`,
`kLCDIFV2_Layer3FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(3)`,
`kLCDIFV2_Layer4FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(4)`,
`kLCDIFV2_Layer5FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(5)`,
`kLCDIFV2_Layer6FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(6)`,
`kLCDIFV2_Layer7FifoEmptyInterrupt` = `LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(7)`,
`kLCDIFV2_Layer0DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(0)`,
`kLCDIFV2_Layer1DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(1)`,
`kLCDIFV2_Layer2DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(2)`,
`kLCDIFV2_Layer3DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(3)`,
`kLCDIFV2_Layer4DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(4)`,
`kLCDIFV2_Layer5DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(5)`,
`kLCDIFV2_Layer6DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(6)`,
`kLCDIFV2_Layer7DmaDoneInterrupt` = `LCDIFV2_MAKE_DMA_DONE_INTERRUPT(7)`,
`kLCDIFV2_Layer0DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(0)`,
`kLCDIFV2_Layer1DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(1)`,
`kLCDIFV2_Layer2DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(2)`,
`kLCDIFV2_Layer3DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(3)`,
`kLCDIFV2_Layer4DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(4)`,
`kLCDIFV2_Layer5DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(5)`,
`kLCDIFV2_Layer6DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(6)`,
`kLCDIFV2_Layer7DmaErrorInterrupt` = `LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(7)`,
`kLCDIFV2_VerticalBlankingInterrupt` = `(1U << 2U)`,
`kLCDIFV2_OutputUnderrunInterrupt` = `(1U << 1U)`,
`kLCDIFV2_VsyncEdgeInterrupt` = `(1U << 0U)` }
The LCDIF v2 interrupts.
- enum `lcdifv2_line_order_t` {


```

kLCDIFV2_LineOrderRGB = 0,
kLCDIFV2_LineOrderRBG,
kLCDIFV2_LineOrderGBR,
kLCDIFV2_LineOrderGRB,
kLCDIFV2_LineOrderBRG,
kLCDIFV2_LineOrderBGR }

```

The LCDIF v2 output line order.

- enum `lcdifv2_csc_mode_t` {
`kLCDIFV2_CscDisable` = 0U,
`kLCDIFV2_CscYUV2RGB`,
`kLCDIFV2_CscYCbCr2RGB` }

LCDIF v2 color space conversion mode.

- enum `lcdifv2_pixel_format_t` {
`kLCDIFV2_PixelFormatIndex1BPP` = `LCDIFV2_CTRLDESCL5_BPP(0U)`,
`kLCDIFV2_PixelFormatIndex2BPP` = `LCDIFV2_CTRLDESCL5_BPP(1U)`,
`kLCDIFV2_PixelFormatIndex4BPP` = `LCDIFV2_CTRLDESCL5_BPP(2U)`,
`kLCDIFV2_PixelFormatIndex8BPP` = `LCDIFV2_CTRLDESCL5_BPP(3U)`,
`kLCDIFV2_PixelFormatRGB565` = `LCDIFV2_CTRLDESCL5_BPP(4U)`,
`kLCDIFV2_PixelFormatARGB1555` = `LCDIFV2_CTRLDESCL5_BPP(5U)`,
`kLCDIFV2_PixelFormatARGB4444` = `LCDIFV2_CTRLDESCL5_BPP(6U)`,
`kLCDIFV2_PixelFormatUYVY`,
`kLCDIFV2_PixelFormatVYUY`,
`kLCDIFV2_PixelFormatYUYV`,
`kLCDIFV2_PixelFormatYVYU`,
`kLCDIFV2_PixelFormatRGB888` = `LCDIFV2_CTRLDESCL5_BPP(8U)`,
`kLCDIFV2_PixelFormatARGB8888` = `LCDIFV2_CTRLDESCL5_BPP(9U)`,
`kLCDIFV2_PixelFormatABGR8888` = `LCDIFV2_CTRLDESCL5_BPP(10U)` }

LCDIF v2 pixel format.

- enum `lcdifv2_alpha_mode_t` {
`kLCDIFV2_AlphaDisable`,
`kLCDIFV2_AlphaOverride`,
`kLCDIFV2_AlphaEmbedded`,
`kLCDIFV2_AlphaPoterDuff` }

LCDIF v2 layer alpha blending mode.

- enum `lcdifv2_pd_alpha_mode_t` {
`kLCDIFV2_PD_AlphaStraight` = 0,
`kLCDIFV2_PD_AlphaInversed` = 1 }

LCDIF v2 PoterDuff alpha mode.

- enum `lcdifv2_pd_color_mode_t` {
`kLCDIFV2_PD_ColorNoAlpha` = 0,
`kLCDIFV2_PD_ColorWithAlpha` = 1 }

LCDIF v2 PoterDuff color mode.

- enum `lcdifv2_pd_global_alpha_mode_t` {
`kLCDIFV2_PD_GlobalAlpha` = 0,
`kLCDIFV2_PD_LocalAlpha` = 1,
`kLCDIFV2_PD_ScaledAlpha` = 2 }

- LCDIF v2 PoterDuff global alpha mode.*
 - enum `lcdifv2_pd_factor_mode_t` {
`kLCDIFV2_PD_FactorOne` = 0,
`kLCDIFV2_PD_FactorZero` = 1,
`kLCDIFV2_PD_FactorStraightAlpha` = 2,
`kLCDIFV2_PD_FactorInversedAlpha` = 3 }

- LCDIF v2 PoterDuff factor mode.*
 - enum `lcdifv2_pd_blend_mode_t` {
`kLCDIFV2_PD_Src` = 0,
`kLCDIFV2_PD_Atop`,
`kLCDIFV2_PD_Over`,
`kLCDIFV2_PD_In`,
`kLCDIFV2_PD_Out`,
`kLCDIFV2_PD_Dst`,
`kLCDIFV2_PD_DstAtop`,
`kLCDIFV2_PD_DstOver`,
`kLCDIFV2_PD_DstIn`,
`kLCDIFV2_PD_DstOut`,
`kLCDIFV2_PD_Xor`,
`kLCDIFV2_PD_Clear`,
`kLCDIFV2_PD_Max` }

- LCDIFv2 Porter Duff blend mode.*
 - enum `lcdifv2_pd_layer_t` {
`kLCDIFV2_PD_SrcLayer` = 0,
`kLCDIFV2_PD_DestLayer` = 1,
`kLCDIFV2_PD_LayerMax` = 2 }

LCDIFv2 Porter Duff layer.

Driver version

- #define `FSL_LCDIFV2_DRIVER_VERSION` (`MAKE_VERSION(2, 2, 3)`)
LCDIF v2 driver version.

LCDIF v2 initialization and de-initialization

- void `LCDIFV2_Init` (`LCDIFV2_Type *base`)
Initializes the LCDIF v2.
- void `LCDIFV2_Deinit` (`LCDIFV2_Type *base`)
Deinitializes the LCDIF peripheral.
- void `LCDIFV2_Reset` (`LCDIFV2_Type *base`)
Reset the LCDIF v2.

Display

- void `LCDIFV2_DisplayGetDefaultConfig` (`lcdifv2_display_config_t *config`)
Gets the LCDIF display default configuration structure.
- void `LCDIFV2_SetDisplayConfig` (`LCDIFV2_Type *base`, const `lcdifv2_display_config_t *config`)
Set the LCDIF v2 display configurations.

- static void [LCDIFV2_EnableDisplay](#) (LCDIFV2_Type *base, bool enable)
Enable or disable the display.

Interrupts

- static void [LCDIFV2_EnableInterrupts](#) (LCDIFV2_Type *base, uint8_t domain, uint32_t mask)
Enables LCDIF interrupt requests.
- static void [LCDIFV2_DisableInterrupts](#) (LCDIFV2_Type *base, uint8_t domain, uint32_t mask)
Disables LCDIF interrupt requests.
- static uint32_t [LCDIFV2_GetInterruptStatus](#) (LCDIFV2_Type *base, uint8_t domain)
Get LCDIF interrupt pending status.
- static void [LCDIFV2_ClearInterruptStatus](#) (LCDIFV2_Type *base, uint8_t domain, uint32_t mask)
Clear LCDIF interrupt pending status.

LUT

- [status_t LCDIFV2_SetLut](#) (LCDIFV2_Type *base, uint8_t layerIndex, const uint32_t *lutData, uint16_t count, bool useShadowLoad)
Set the LUT data.

Layer operation

- static void [LCDIFV2_SetLayerSize](#) (LCDIFV2_Type *base, uint8_t layerIndex, uint16_t width, uint16_t height)
Set the layer dimension.
- static void [LCDIFV2_SetLayerOffset](#) (LCDIFV2_Type *base, uint8_t layerIndex, uint16_t offsetX, uint16_t offsetY)
Set the layer position in output frame.
- void [LCDIFV2_SetLayerBufferConfig](#) (LCDIFV2_Type *base, uint8_t layerIndex, const [lcdifv2_buffer_config_t](#) *config)
Set the layer source buffer configuration.
- static void [LCDIFV2_SetLayerBufferAddr](#) (LCDIFV2_Type *base, uint8_t layerIndex, uint32_t addr)
Set the layer source buffer address.
- static void [LCDIFV2_EnableLayer](#) (LCDIFV2_Type *base, uint8_t layerIndex, bool enable)
Enable or disable the layer.
- static void [LCDIFV2_TriggerLayerShadowLoad](#) (LCDIFV2_Type *base, uint8_t layerIndex)
Trigger the layer configuration shadow load.
- static void [LCDIFV2_SetLayerBackgroundColor](#) (LCDIFV2_Type *base, uint8_t layerIndex, uint32_t backgroundColor)
Set the layer back ground color.
- void [LCDIFV2_SetLayerBlendConfig](#) (LCDIFV2_Type *base, uint8_t layerIndex, const [lcdifv2_blend_config_t](#) *config)
Set the layer alpha blend mode.
- void [LCDIFV2_SetCscMode](#) (LCDIFV2_Type *base, uint8_t layerIndex, [lcdifv2_csc_mode_t](#) mode)
Set the color space conversion mode.

Porter Duff

- [status_t LCDIFV2_GetPorterDuffConfig](#) ([lcdifv2_pd_blend_mode_t](#) mode, [lcdifv2_pd_layer_t](#) layer, [lcdifv2_blend_config_t](#) *config)
Get the blend configuration for Porter Duff blend.

38.4 Data Structure Documentation

38.4.1 struct lcdifv2_display_config_t

Data Fields

- [uint16_t panelWidth](#)
Display panel width, pixels per line.
- [uint16_t panelHeight](#)
Display panel height, how many lines per panel.
- [uint8_t hsw](#)
HSYNC pulse width.
- [uint8_t hfp](#)
Horizontal front porch.
- [uint8_t hbp](#)
Horizontal back porch.
- [uint8_t vsw](#)
VSYNC pulse width.
- [uint8_t vfp](#)
Vertical front porch.
- [uint8_t vbp](#)
Vertical back porch.
- [uint32_t polarityFlags](#)
OR'ed value of [_lcdifv2_polarity_flags](#), used to control the signal polarity.
- [lcdifv2_line_order_t lineOrder](#)
Line order.

Field Documentation

- (1) [uint16_t lcdifv2_display_config_t::panelWidth](#)
- (2) [uint16_t lcdifv2_display_config_t::panelHeight](#)
- (3) [uint8_t lcdifv2_display_config_t::hsw](#)
- (4) [uint8_t lcdifv2_display_config_t::hfp](#)
- (5) [uint8_t lcdifv2_display_config_t::hbp](#)
- (6) [uint8_t lcdifv2_display_config_t::vsw](#)
- (7) [uint8_t lcdifv2_display_config_t::vfp](#)
- (8) [uint8_t lcdifv2_display_config_t::vbp](#)

(9) `uint32_t lcdifv2_display_config_t::polarityFlags`

(10) `lcdifv2_line_order_t lcdifv2_display_config_t::lineOrder`

38.4.2 struct `lcdifv2_buffer_config_t`

Data Fields

- `uint16_t strideBytes`
Number of bytes between two vertically adjacent pixels, suggest 64-bit aligned.
- `lcdifv2_pixel_format_t pixelFormat`
Source buffer pixel format.

Field Documentation

(1) `uint16_t lcdifv2_buffer_config_t::strideBytes`

(2) `lcdifv2_pixel_format_t lcdifv2_buffer_config_t::pixelFormat`

38.4.3 struct `lcdifv2_blend_config_t`

Data Fields

- `uint8_t globalAlpha`
Global alpha value, only used when `alphaMode` is `kLCDIFV2_AlphaOverride` or `kLCDIFV2_AlphaPoterDuff`.
- `lcdifv2_alpha_mode_t alphaMode`
Alpha mode.
- `lcdifv2_pd_alpha_mode_t pdAlphaMode`
PoterDuff alpha mode, only used when `alphaMode` is `kLCDIFV2_AlphaPoterDuff`.
- `lcdifv2_pd_color_mode_t pdColorMode`
PoterDuff color mode, only used when `alphaMode` is `kLCDIFV2_AlphaPoterDuff`.
- `lcdifv2_pd_global_alpha_mode_t pdGlobalAlphaMode`
PoterDuff global alpha mode, only used when @ref `alphaMode` is `kLCDIFV2_AlphaPoterDuff`
- `lcdifv2_pd_factor_mode_t pdFactorMode`
PoterDuff factor mode, only used when @ref `alphaMode` is @ref `kLCDIFV2_AlphaPoterDuff`

Field Documentation

(1) `lcdifv2_alpha_mode_t lcdifv2_blend_config_t::alphaMode`

38.5 Macro Definition Documentation

38.5.1 `#define LCDIFV2_MAKE_FIFO_EMPTY_INTERRUPT(layer) (1U << ((uint32_t)(layer) + 24U))`

38.5.2 `#define LCDIFV2_MAKE_DMA_DONE_INTERRUPT(layer) (1U << ((uint32_t)(layer) + 16U))`

38.5.3 `#define LCDIFV2_MAKE_DMA_ERROR_INTERRUPT(layer) (1U << ((uint32_t)(layer) + 8U))`

38.6 Enumeration Type Documentation

38.6.1 `enum _lcdifv2_polarity_flags`

Enumerator

kLCDIFV2_VsyncActiveLow VSYNC active low.
kLCDIFV2_VsyncActiveHigh VSYNC active high.
kLCDIFV2_HsyncActiveLow HSYNC active low.
kLCDIFV2_HsyncActiveHigh HSYNC active high.
kLCDIFV2_DataEnableActiveLow Data enable line active low.
kLCDIFV2_DataEnableActiveHigh Data enable line active high.
kLCDIFV2_DriveDataOnFallingClkEdge Output data on falling clock edge, capture data on rising clock edge.
kLCDIFV2_DriveDataOnRisingClkEdge Output data on rising clock edge, capture data on falling clock edge.
kLCDIFV2_DataActiveLow Data active high.
kLCDIFV2_DataActiveHigh Data active high.

38.6.2 `enum _lcdifv2_interrupt`

Enumerator

kLCDIFV2_Layer0FifoEmptyInterrupt Layer 0 FIFO empty.
kLCDIFV2_Layer1FifoEmptyInterrupt Layer 1 FIFO empty.
kLCDIFV2_Layer2FifoEmptyInterrupt Layer 2 FIFO empty.
kLCDIFV2_Layer3FifoEmptyInterrupt Layer 3 FIFO empty.
kLCDIFV2_Layer4FifoEmptyInterrupt Layer 4 FIFO empty.
kLCDIFV2_Layer5FifoEmptyInterrupt Layer 5 FIFO empty.
kLCDIFV2_Layer6FifoEmptyInterrupt Layer 6 FIFO empty.
kLCDIFV2_Layer7FifoEmptyInterrupt Layer 7 FIFO empty.
kLCDIFV2_Layer0DmaDoneInterrupt Layer 0 DMA done.
kLCDIFV2_Layer1DmaDoneInterrupt Layer 1 DMA done.
kLCDIFV2_Layer2DmaDoneInterrupt Layer 2 DMA done.
kLCDIFV2_Layer3DmaDoneInterrupt Layer 3 DMA done.
kLCDIFV2_Layer4DmaDoneInterrupt Layer 4 DMA done.
kLCDIFV2_Layer5DmaDoneInterrupt Layer 5 DMA done.
kLCDIFV2_Layer6DmaDoneInterrupt Layer 6 DMA done.

kLCDIFV2_Layer7DmaDoneInterrupt Layer 7 DMA done.
kLCDIFV2_Layer0DmaErrorInterrupt Layer 0 DMA error.
kLCDIFV2_Layer1DmaErrorInterrupt Layer 1 DMA error.
kLCDIFV2_Layer2DmaErrorInterrupt Layer 2 DMA error.
kLCDIFV2_Layer3DmaErrorInterrupt Layer 3 DMA error.
kLCDIFV2_Layer4DmaErrorInterrupt Layer 4 DMA error.
kLCDIFV2_Layer5DmaErrorInterrupt Layer 5 DMA error.
kLCDIFV2_Layer6DmaErrorInterrupt Layer 6 DMA error.
kLCDIFV2_Layer7DmaErrorInterrupt Layer 7 DMA error.
kLCDIFV2_VerticalBlankingInterrupt Start of vertical blanking period.
kLCDIFV2_OutputUnderrunInterrupt Output buffer underrun.
kLCDIFV2_VsyncEdgeInterrupt Interrupt at VSYNC edge.

38.6.3 enum lcdifv2_line_order_t

Enumerator

kLCDIFV2_LineOrderRGB RGB.
kLCDIFV2_LineOrderRBG RBG.
kLCDIFV2_LineOrderGBR GBR.
kLCDIFV2_LineOrderGRB GRB.
kLCDIFV2_LineOrderBRG BRG.
kLCDIFV2_LineOrderBGR BGR.

38.6.4 enum lcdifv2_csc_mode_t

Enumerator

kLCDIFV2_CscDisable Disable the CSC.
kLCDIFV2_CscYUV2RGB YUV to RGB.
kLCDIFV2_CscYCbCr2RGB YCbCr to RGB.

38.6.5 enum lcdifv2_pixel_format_t

Enumerator

kLCDIFV2_PixelFormatIndex1BPP LUT index 1 bit.
kLCDIFV2_PixelFormatIndex2BPP LUT index 2 bit.
kLCDIFV2_PixelFormatIndex4BPP LUT index 4 bit.
kLCDIFV2_PixelFormatIndex8BPP LUT index 8 bit.
kLCDIFV2_PixelFormatRGB565 RGB565, two pixels use 32 bits.

kLCDIFV2_PixelFormatARGB1555 ARGB1555, two pixels use 32 bits.
kLCDIFV2_PixelFormatARGB4444 ARGB4444, two pixels use 32 bits.
kLCDIFV2_PixelFormatUYVY UYVY, only layer 0 and layer 1 support this.
kLCDIFV2_PixelFormatVYUY VYUY, only layer 0 and layer 1 support this.
kLCDIFV2_PixelFormatYUYV YUYV, only layer 0 and layer 1 support this.
kLCDIFV2_PixelFormatYVYU YVYU, only layer 0 and layer 1 support this.
kLCDIFV2_PixelFormatRGB888 RGB888 packed, one pixel uses 24 bits.
kLCDIFV2_PixelFormatARGB8888 ARGB8888 unpacked, one pixel uses 32 bits.
kLCDIFV2_PixelFormatABGR8888 ABGR8888 unpacked, one pixel uses 32 bits.

38.6.6 enum lcdifv2_alpha_mode_t

Enumerator

kLCDIFV2_AlphaDisable Disable alpha blend.
kLCDIFV2_AlphaOverride Use the global alpha value, pixel defined alpha value is overridden.
kLCDIFV2_AlphaEmbedded Use the pixel defined alpha value.
kLCDIFV2_AlphaPoterDuff Use the PoterDuff alpha blending.

38.6.7 enum lcdifv2_pd_alpha_mode_t

Enumerator

kLCDIFV2_PD_AlphaStraight Straight mode.
kLCDIFV2_PD_AlphaInversed Inversed mode.

38.6.8 enum lcdifv2_pd_color_mode_t

Enumerator

kLCDIFV2_PD_ColorNoAlpha Output color directly.
kLCDIFV2_PD_ColorWithAlpha Output color multiples alpha.

38.6.9 enum lcdifv2_pd_global_alpha_mode_t

Enumerator

kLCDIFV2_PD_GlobalAlpha Use global alpha.
kLCDIFV2_PD_LocalAlpha Use local alpha.
kLCDIFV2_PD_ScaledAlpha Use scaled alpha.

38.6.10 enum lcdifv2_pd_factor_mode_t

Enumerator

kLCDIFV2_PD_FactorOne Use 1.
kLCDIFV2_PD_FactorZero Use 0.
kLCDIFV2_PD_FactorStraightAlpha Use straight alpha.
kLCDIFV2_PD_FactorInversedAlpha Use inversed alpha.

38.6.11 enum lcdifv2_pd_blend_mode_t

Note: Don't change the enum item value

Enumerator

kLCDIFV2_PD_Src Source Only.
kLCDIFV2_PD_Atop Source Atop.
kLCDIFV2_PD_Over Source Over.
kLCDIFV2_PD_In Source In.
kLCDIFV2_PD_Out Source Out.
kLCDIFV2_PD_Dst Destination Only.
kLCDIFV2_PD_DstAtop Destination Atop.
kLCDIFV2_PD_DstOver Destination Over.
kLCDIFV2_PD_DstIn Destination In.
kLCDIFV2_PD_DstOut Destination Out.
kLCDIFV2_PD_Xor XOR.
kLCDIFV2_PD_Clear Clear.
kLCDIFV2_PD_Max Used for boarder detection.

38.6.12 enum lcdifv2_pd_layer_t

Note: Don't change the enum item value

Enumerator

kLCDIFV2_PD_SrcLayer Source layer.
kLCDIFV2_PD_DestLayer Destination layer.
kLCDIFV2_PD_LayerMax Used for boarder detection.

38.7 Function Documentation

38.7.1 void LCDIFV2_Init (LCDIFV2_Type * *base*)

This function ungates the LCDIF v2 clock and release the peripheral reset.

Parameters

<i>base</i>	LCDIF v2 peripheral base address.
-------------	-----------------------------------

38.7.2 void LCDIFV2_Deinit (LCDIFV2_Type * *base*)

Parameters

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

38.7.3 void LCDIFV2_Reset (LCDIFV2_Type * *base*)

Parameters

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

38.7.4 void LCDIFV2_DisplayGetDefaultConfig (lcdifv2_display_config_t * *config*)

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

```

config->panelWidth      = 0U;
config->panelHeight     = 0U;
config->hsw              = 3U;
config->hfp              = 3U;
config->hbp              = 3U;
config->vsw              = 3U;
config->vfp              = 3U;
config->vbp              = 3U;
config->polarityFlags = kLCDIFV2_VsyncActiveHigh |
                        kLCDIFV2_HsyncActiveHigh | kLCDIFV2_DataEnableActiveHigh
                        |
                        kLCDIFV2_DriveDataOnRisingClkEdge |
                        kLCDIFV2_DataActiveHigh;
config->lineOrder         = kLCDIFV2_LineOrderRGB;

```

Parameters

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

38.7.5 void LCDIFV2_SetDisplayConfig (LCDIFV2_Type * *base*, const lcdifv2_display_config_t * *config*)

Parameters

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

38.7.6 static void LCDIFV2_EnableDisplay (LCDIFV2_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	LCDIF peripheral base address.
<i>enable</i>	Enable or disable.

38.7.7 static void LCDIFV2_EnableInterrupts (LCDIFV2_Type * *base*, uint8_t *domain*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	LCDIF peripheral base address.
<i>domain</i>	CPU domain the interrupt signal routed to.
<i>mask</i>	interrupt source, OR'ed value of _lcdifv2_interrupt.

38.7.8 static void LCDIFV2_DisableInterrupts (LCDIFV2_Type * *base*, uint8_t *domain*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	LCDIF peripheral base address.
<i>domain</i>	CPU domain the interrupt signal routed to.
<i>mask</i>	interrupt source, OR'ed value of _lcdifv2_interrupt.

38.7.9 static uint32_t LCDIFV2_GetInterruptStatus (LCDIFV2_Type * *base*, uint8_t *domain*) [inline], [static]

Parameters

<i>base</i>	LCDIF peripheral base address.
<i>domain</i>	CPU domain the interrupt signal routed to.

Returns

Interrupt pending status, OR'ed value of _lcdifv2_interrupt.

38.7.10 static void LCDIFV2_ClearInterruptStatus (LCDIFV2_Type * *base*, uint8_t *domain*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	LCDIF peripheral base address.
<i>domain</i>	CPU domain the interrupt signal routed to.
<i>mask</i>	of the flags to clear, OR'ed value of _lcdifv2_interrupt.

38.7.11 status_t LCDIFV2_SetLut (LCDIFV2_Type * *base*, uint8_t *layerIndex*, const uint32_t * *lutData*, uint16_t *count*, bool *useShadowLoad*)

This function sets the specific layer LUT data, if *useShadowLoad* is true, call [LCDIFV2_TriggerLayerShadowLoad](#) after this function, the LUT will be loaded to the hardware during next vertical blanking period. If *useShadowLoad* is false, the LUT data is loaded to hardware directly.

Parameters

<i>base</i>	LCDIF v2 peripheral base address.
<i>layerIndex</i>	Which layer to set.
<i>lutData</i>	The LUT data to load.
<i>count</i>	Count of <code>lutData</code> .
<i>useShadow-Load</i>	Use shadow load.

Return values

<i>kStatus_Success</i>	Set success.
<i>kStatus_Fail</i>	Previous LUT data is not loaded to hardware yet.

38.7.12 `static void LCDIFV2_SetLayerSize (LCDIFV2_Type * base, uint8_t layerIndex, uint16_t width, uint16_t height) [inline], [static]`

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Layer <code>layerIndex</code> .
<i>width</i>	Layer width in pixel.
<i>height</i>	Layer height.

Note

The layer width must be in multiples of the number of pixels that can be stored in 32 bits

38.7.13 `static void LCDIFV2_SetLayerOffset (LCDIFV2_Type * base, uint8_t layerIndex, uint16_t offsetX, uint16_t offsetY) [inline], [static]`

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Layer layerIndex.
<i>offsetX</i>	Horizontal offset, start from 0.
<i>offsetY</i>	Vertical offset, start from 0.

38.7.14 void LCDIFV2_SetLayerBufferConfig (LCDIFV2_Type * *base*, uint8_t *layerIndex*, const lcdifv2_buffer_config_t * *config*)

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Layer layerIndex.
<i>config</i>	Pointer to the configuration.

38.7.15 static void LCDIFV2_SetLayerBufferAddr (LCDIFV2_Type * *base*, uint8_t *layerIndex*, uint32_t *addr*) [inline], [static]

This function is used for fast runtime source buffer change.

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Layer layerIndex.
<i>addr</i>	The new source buffer address passed to the layer, should be 64-bit aligned.

38.7.16 static void LCDIFV2_EnableLayer (LCDIFV2_Type * *base*, uint8_t *layerIndex*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
-------------	----------------------------------

<i>layerIndex</i>	Layer layerIndex.
<i>enable</i>	Pass in true to enable, false to disable.

38.7.17 static void LCDIFV2_TriggerLayerShadowLoad (LCDIFV2_Type * *base*, uint8_t *layerIndex*) [inline], [static]

The new layer configurations are written to the shadow registers first, When all configurations written finished, call this function, then shadowed control registers are updated to the active control registers on VSYNC of next frame.

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Layer layerIndex.

38.7.18 static void LCDIFV2_SetLayerBackGroundColor (LCDIFV2_Type * *base*, uint8_t *layerIndex*, uint32_t *backGroundColor*) [inline], [static]

The back ground color is used when layer not activated.

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Index of the layer.
<i>backGround-Color</i>	Background color to use when this layer is not active.

38.7.19 void LCDIFV2_SetLayerBlendConfig (LCDIFV2_Type * *base*, uint8_t *layerIndex*, const lcdifv2_blend_config_t * *config*)

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
-------------	----------------------------------

<i>layerIndex</i>	Index of the CSC unit.
<i>config</i>	Pointer to the blend configuration.

38.7.20 void LCDIFV2_SetCscMode (LCDIFV2_Type * *base*, uint8_t *layerIndex*, lcdifv2_csc_mode_t *mode*)

Supports YUV2RGB and YCbCr2RGB.

Parameters

<i>base</i>	LCDIFv2 peripheral base address.
<i>layerIndex</i>	Index of the layer.
<i>mode</i>	The conversion mode.

38.7.21 status_t LCDIFV2_GetPorterDuffConfig (lcdifv2_pd_blend_mode_t *mode*, lcdifv2_pd_layer_t *layer*, lcdifv2_blend_config_t * *config*)

This function gets the blend configuration for Porter Duff blend, config->pdFactorMode is set according to *layer* and *mode*, other blend configurations are set to:

```
config->pdAlphaMode = kLCDIFV2_PD_AlphaStraight;
config->pdColorMode = kLCDIFV2_PD_ColorStraight;
config->pdGlobalAlphaMode = kLCDIFV2_PD_LocalAlpha;
config->alphaMode = kLCDIFV2_AlphaPorterDuff;
```

This is the basic Porter Duff blend configuration, user still could modify the configurations after this function.

Parameters

<i>mode</i>	Porter Duff blend mode.
<i>layer</i>	The configuration for source layer or destination layer.
<i>config</i>	Pointer to the configuration.

Return values

<i>kStatus_Success</i>	Get the configuration successfully.
<i>kStatus_InvalidArgument</i>	The argument is invalid.

Chapter 39

LPADC: 12-bit SAR Analog-to-Digital Converter Driver

39.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit SAR Analog-to-Digital Converter (LP-ADC) module of MCUXpresso SDK devices.

39.2 Typical use case

39.2.1 Polling Configuration

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

39.2.2 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...](#)

Macros

- #define [LPADC_GET_ACTIVE_COMMAND_STATUS](#)(statusVal) ((statusVal & ADC_STAT_CMDACT_MASK) >> ADC_STAT_CMDACT_SHIFT)
Define the MACRO function to get command status from status value.
- #define [LPADC_GET_ACTIVE_TRIGGER_STATUE](#)(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_ResultFIFOOverflowFlag` = `ADC_STAT_FOF_MASK`,
`kLPADC_ResultFIFOReadyFlag` = `ADC_STAT_RDY_MASK` }
Define hardware flags of the module.
- enum `_lpadc_interrupt_enable` {
`kLPADC_ResultFIFOOverflowInterruptEnable` = `ADC_IE_FOFIE_MASK`,
`kLPADC_FIFOWatermarkInterruptEnable` = `ADC_IE_FWMIE_MASK` }
Define interrupt switchers of the module.
- 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` = `0U`,
`kLPADC_SampleChannelSingleEndSideB` = `1U`,
`kLPADC_SampleChannelDiffBothSideAB` = `2U`,
`kLPADC_SampleChannelDiffBothSideBA` = `3U` }
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` }
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_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_trigger_priority_policy_t {
    kLPADC_TriggerPriorityPreemptImmediately = 0U,
    kLPADC_TriggerPriorityPreemptSoftly = 1U,
    kLPADC_TriggerPriorityPreemptSubsequently = 2U }
    Define enumeration of trigger priority policy.

```

Driver version

- #define **FSL_LPADC_DRIVER_VERSION** (MAKE_VERSION(2, 5, 1))
LPADC driver version 2.5.1.

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_DoResetFIFO** (ADC_Type *base)
Do reset the conversion FIFO.
- 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.

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_EnableFIFOWatermarkDMA](#) (ADC_Type *base, bool enable)
Switch on/off the DMA trigger for FIFO watermark event.

Trigger and conversion with FIFO.

- static uint32_t [LPADC_GetConvResultCount](#) (ADC_Type *base)
Get the count of result kept in conversion FIFO.
- bool [LPADC_GetConvResult](#) (ADC_Type *base, [lpadc_conv_result_t](#) *result)
Get the result in conversion FIFO.
- 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.

39.3 Data Structure Documentation

39.3.1 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.
- 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_power_level_mode_t](#) [powerLevelMode](#)
Power Configuration Selection.
- [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 FIFOWatermark`
FIFOWatermark is a programmable threshold setting.

Field Documentation

(1) `bool lpadc_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) `bool lpadc_config_t::enableAnalogPreliminary`

(3) `uint32_t lpadc_config_t::powerUpDelay`

The startup delay count of $(\text{powerUpDelay} * 4)$ ADCK cycles must result in a longer delay than the analog startup time.

(4) `lpadc_reference_voltage_source_t lpadc_config_t::referenceVoltageSource`

(5) `lpadc_power_level_mode_t lpadc_config_t::powerLevelMode`

(6) `lpadc_trigger_priority_policy_t lpadc_config_t::triggerPriorityPolicy`

(7) `bool lpadc_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 lpadc_config_t::convPauseDelay`

The pause delay is a count of $(\text{convPauseDelay} * 4)$ ADCK cycles. Only available when ADC pausing function is enabled. The available value range is in 9-bit.

(9) `uint32_t lpadc_config_t::FIFOWatermark`

When the number of datawords stored in the ADC Result FIFO is greater than the value in this field, the ready flag would be asserted to indicate stored data has reached the programmable threshold.

39.3.2 `struct lpadc_conv_command_config_t`

Data Fields

- `lpadc_sample_scale_mode_t sampleScaleMode`
Sample scale mode.

- `lpadc_sample_channel_mode_t` `sampleChannelMode`
Channel sample mode.
- `uint32_t` `channelNumber`
Channel number, select the channel or channel pair.
- `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.

Field Documentation

(1) `lpadc_sample_scale_mode_t` `lpadc_conv_command_config_t::sampleScaleMode`

(2) `lpadc_sample_channel_mode_t` `lpadc_conv_command_config_t::sampleChannelMode`

(3) `uint32_t` `lpadc_conv_command_config_t::channelNumber`

(4) `uint32_t` `lpadc_conv_command_config_t::chainedNextCommandNumber`

1-15 is available, 0 is to terminate the chain after this command.

(5) `bool` `lpadc_conv_command_config_t::enableAutoChannelIncrement`

(6) `uint32_t` `lpadc_conv_command_config_t::loopCount`

Command executes LOOP+1 times. 0-15 is available.

(7) `lpadc_hardware_average_mode_t` `lpadc_conv_command_config_t::hardwareAverageMode`

(8) `lpadc_sample_time_mode_t` `lpadc_conv_command_config_t::sampleTimeMode`

(9) `lpadc_hardware_compare_mode_t` `lpadc_conv_command_config_t::hardwareCompareMode`

(10) `uint32_t` `lpadc_conv_command_config_t::hardwareCompareValueHigh`

The available value range is in 16-bit.

(11) uint32_t lpadc_conv_command_config_t::hardwareCompareValueLow

The available value range is in 16-bit.

39.3.3 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](#)
Sets the priority of the associated trigger source.
- bool [enableHardwareTrigger](#)
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.

39.3.4 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

39.4 Macro Definition Documentation

39.4.1 #define FSL_LPADC_DRIVER_VERSION (MAKE_VERSION(2, 5, 1))

39.4.2 #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\(\)](#).

39.4.3 #define LPADC_GET_ACTIVE_TRIGGER_STATUE(*statusVal*) ((statusVal & ADC_STAT_TRGACT_MASK) >> ADC_STAT_TRGACT_SHIFT)

The statusVal is the return value from [LPADC_GetStatusFlags\(\)](#).

39.5 Enumeration Type Documentation

39.5.1 enum _lpadc_status_flags

Enumerator

kLPADC_ResultFIFOOverflowFlag Indicates that more data has been written to the Result FIFO than it can hold.

kLPADC_ResultFIFOReadyFlag Indicates when the number of valid datawords in the result FIFO is greater than the setting watermark level.

39.5.2 enum _lpadc_interrupt_enable

Enumerator

kLPADC_ResultFIFOOverflowInterruptEnable Configures ADC to generate overflow interrupt requests when FOF flag is asserted.

kLPADC_FIFOWatermarkInterruptEnable Configures ADC to generate watermark interrupt requests when RDY flag is asserted.

39.5.3 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. (Factor of 30/64).
kLPADC_SampleFullScale Full scale (Factor of 1).

39.5.4 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, using side A.
kLPADC_SampleChannelSingleEndSideB Single end mode, using side B.
kLPADC_SampleChannelDiffBothSideAB Differential mode, using A as plus side and B as minus side.
kLPADC_SampleChannelDiffBothSideBA Differential mode, using B as plus side and A as minus side.

39.5.5 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.

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.

39.5.6 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.

39.5.7 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.

39.5.8 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.

39.5.9 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.

39.5.10 enum lpadc_trigger_priority_policy_t

This selection controls how higher priority triggers are handled.

Enumerator

- kLPADC_TriggerPriorityPreemptImmediately*** 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.
- kLPADC_TriggerPriorityPreemptSoftly*** 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.
- kLPADC_TriggerPriorityPreemptSubsequently*** 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.

39.6 Function Documentation

39.6.1 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".

39.6.2 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.
---------------	-------------------------------------

39.6.3 void LPADC_Deinit (ADC_Type * *base*)

Parameters

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

39.6.4 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.

39.6.5 static void LPADC_DoResetFIFO (ADC_Type * *base*) [inline], [static]

Parameters

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

39.6.6 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.
-------------	--------------------------------

39.6.7 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](#).

39.6.8 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 _lpadc_status_flags .

39.6.9 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 _lpadc_interrupt_enable .

39.6.10 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 _lpadc_interrupt_enable .

39.6.11 static void LPADC_EnableFIFOWatermarkDMA (ADC_Type * *base*, bool *enable*) [inline], [static]

Parameters

<i>base</i>	LPADC peripheral base address.
<i>enable</i>	Switcher to the event.

39.6.12 static uint32_t LPADC_GetConvResultCount (ADC_Type * *base*) [inline], [static]

Parameters

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

Returns

The count of result kept in conversion FIFO.

39.6.13 bool LPADC_GetConvResult (ADC_Type * *base*, lpadc_conv_result_t * *result*)

Parameters

<i>base</i>	LPADC peripheral base address.
<i>result</i>	Pointer to structure variable that keeps the conversion result in conversion FIFO.

Returns

Status whether FIFO entry is valid.

39.6.14 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 lpadc_conv_trigger_config_t .

39.6.15 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->commandIdSource      = 0U;
* config->loopCountIndex      = 0U;
* config->triggerIdSource      = 0U;
* config->enableHardwareTrigger = false;
*
```

Parameters

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

39.6.16 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.

39.6.17 void LPADC_SetConvCommandConfig (ADC_Type * *base*, uint32_t *commandId*, const lpadc_conv_command_config_t * *config*)

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 lpadc_conv_command_config_t .

39.6.18 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->channelSampleMode         = kLPADC_SampleChannelSingleEndSideA
*
* config->channelNumber              = 0U;
* config->chainedNextCmdNumber       = 0U;
* config->enableAutoChannelIncrement = false;
* config->loopCount                  = 0U;
* config->hardwareAverageMode        = kLPADC_HardwareAverageCount1;
* config->sampleTimeMode             = kLPADC_SampleTimeADCK3;
* config->hardwareCompareMode        = kLPADC_HardwareCompareDisabled;
* config->hardwareCompareValueHigh   = 0U;
* config->hardwareCompareValueLow    = 0U;
* config->conversionResolutionMode   = kLPADC_ConversionResolutionStandard;
* config->enableWaitTrigger           = false;
*
```

Parameters

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

Chapter 40

LPI2C: Low Power Inter-Integrated Circuit Driver

40.1 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.

Driver version

- `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`
LPI2C driver version.

40.2 Macro Definition Documentation

40.2.1 `#define FSL_LPI2C_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))`

40.2.2 #define I2C_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

40.3 Enumeration Type Documentation

40.3.1 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.

40.4 LPI2C Master Driver

40.4.1 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)

Configures LPI2C master data match feature.

- [status_t LPI2C_MasterCheckAndClearError](#) (LPI2C_Type *base, uint32_t status)
- [status_t LPI2C_CheckForBusyBus](#) (LPI2C_Type *base)
- 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)
 - static void [LPI2C_MasterClearStatusFlags](#) (LPI2C_Type *base, uint32_t statusMask)
- Gets the LPI2C master status flags.*
Clears the LPI2C master status flag state.

Interrupts

- static void [LPI2C_MasterEnableInterrupts](#) (LPI2C_Type *base, uint32_t interruptMask)
 - static void [LPI2C_MasterDisableInterrupts](#) (LPI2C_Type *base, uint32_t interruptMask)
 - static uint32_t [LPI2C_MasterGetEnabledInterrupts](#) (LPI2C_Type *base)
- Enables the LPI2C master interrupt requests.*
Disables the LPI2C master interrupt requests.
Returns the set of currently enabled LPI2C master interrupt requests.

DMA control

- static void [LPI2C_MasterEnableDMA](#) (LPI2C_Type *base, bool enableTx, bool enableRx)
 - static uint32_t [LPI2C_MasterGetTxFifoAddress](#) (LPI2C_Type *base)
 - static uint32_t [LPI2C_MasterGetRxFifoAddress](#) (LPI2C_Type *base)
- Enables or disables LPI2C master DMA requests.*
Gets LPI2C master transmit data register address for DMA transfer.
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)
 - static void [LPI2C_MasterGetFifoCounts](#) (LPI2C_Type *base, size_t *rxCount, size_t *txCount)
- Sets the watermarks for LPI2C master FIFOs.*
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 baud-Rate_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.

40.4.2 Data Structure Documentation

40.4.2.1 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`

40.4.2.2 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`

40.4.2.3 struct `_lpi2c_master_transfer`

This structure is used to pass transaction parameters to the `LPI2C_MasterTransferNonBlocking()` API.

Data Fields

- `uint32_t flags`

- `uint16_t slaveAddress`
Bit mask of options for the transfer.
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_TransferDefaultFlag` 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`

40.4.2.4 `struct _lpi2c_master_handle`

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

40.4.3 Typedef Documentation

40.4.3.1 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>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.
-----------------	--

40.4.4 Enumeration Type Documentation

40.4.4.1 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.

40.4.4.2 enum lpi2c_direction_t

Enumerator

kLPI2C_Write Master transmit.
kLPI2C_Read Master receive.

40.4.4.3 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)

40.4.4.4 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.

40.4.4.5 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.

40.4.4.6 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.

40.4.4.7 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.

40.4.5 Function Documentation

40.4.5.1 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       = 100000U;
* 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

out	<i>masterConfig</i>	User provided configuration structure for default values. Refer to lpi2c-_master_config_t .
-----	---------------------	---

40.4.5.2 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 LPI2C_MasterGetDefaultConfig() 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.

40.4.5.3 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.
-------------	------------------------------------

40.4.5.4 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.

40.4.5.5 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.
-------------	------------------------------------

40.4.5.6 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.

40.4.5.7 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](#)

40.4.5.8 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 <code>_lpi2c_master_flags</code> enumerators OR'd together. You may pass the result of a previous call to LPI2C_MasterGetStatusFlags() .

See Also

[_lpi2c_master_flags](#).

40.4.5.9 **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 <code>_lpi2c_master_flags</code> for the set of constants that should be OR'd together to form the bit mask.

40.4.5.10 **static void LPI2C_MasterDisableInterrupts (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 disable. See <code>_lpi2c_master_flags</code> for the set of constants that should be OR'd together to form the bit mask.

40.4.5.11 **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.

40.4.5.12 `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.

40.4.5.13 `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.

40.4.5.14 `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.

40.4.5.15 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 kLPI2C_MasterTxReadyFlag 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 kLPI2C_MasterRxReadyFlag 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.

40.4.5.16 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.

40.4.5.17 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.

40.4.5.18 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.

40.4.5.19 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 kLPI2C_Read or kLPI2C_Write . 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.

40.4.5.20 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 kLPI2C_Read or kLPI2C_Write . 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.

40.4.5.21 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.

40.4.5.22 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.

40.4.5.23 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.

40.4.5.24 status_t LPI2C_MasterTransferBlocking (LPI2C_Type * *base*, lpi2c_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.

**40.4.5.25 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.

**40.4.5.26 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.

**40.4.5.27 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_NoTransferInProgress</i>	There is not a non-blocking transaction currently in progress.

40.4.5.28 void LPI2C_MasterTransferAbort (LPI2C_Type * *base*, lpi2c_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.

Return values

<i>kStatus_Success</i>	A transaction was successfully aborted.
<i>kStatus_LPI2C_Idle</i>	There is not a non-blocking transaction currently in progress.

40.4.5.29 void LPI2C_MasterTransferHandleIRQ (LPI2C_Type * *base*, void * *lpi2cMasterHandle*)

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>lpi2cMaster-Handle</i>	Pointer to the LPI2C master driver handle.

40.5 LPI2C Slave Driver

40.5.1 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 enable-Rx, 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 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.

40.5.2 Data Structure Documentation

40.5.2.1 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.
- 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.
- bool [enableAck](#)
Enables SCL clock stretching during slave-transmit address byte(s) and slave-receiver address and data byte(s) to allow software to write the Transmit ACK Register before the ACK or NACK is transmitted.
- 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.

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) **bool** lpi2c_slave_config_t::ignoreAck

(13) **bool** lpi2c_slave_config_t::enableReceivedAddressRead

(14) **uint32_t** lpi2c_slave_config_t::sdaGlitchFilterWidth_ns

Set to 0 to disable.

(15) **uint32_t** lpi2c_slave_config_t::sclGlitchFilterWidth_ns

Set to 0 to disable.

(16) **uint32_t** lpi2c_slave_config_t::dataValidDelay_ns

(17) **uint32_t** lpi2c_slave_config_t::clockHoldTime_ns

40.5.2.2 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`

40.5.2.3 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`

40.5.3 Typedef Documentation

40.5.3.1 `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.

40.5.4 Enumeration Type Documentation

40.5.4.1 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.

40.5.4.2 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.

40.5.4.3 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.

40.5.5 Function Documentation

40.5.5.1 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

out	<i>slaveConfig</i>	User provided configuration structure that is set to default values. Refer to lpi2c_slave_config_t .
-----	--------------------	--

40.5.5.2 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 LPI2C_SlaveGetDefaultConfig() 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.

40.5.5.3 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.
-------------	------------------------------------

40.5.5.4 static void LPI2C_SlaveReset (LPI2C_Type * *base*) [inline], [static]

Parameters

<i>base</i>	The LPI2C peripheral base address.
-------------	------------------------------------

40.5.5.5 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.

40.5.5.6 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](#)

40.5.5.7 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 _lpi2c_slave_flags enumerators OR'd together. You may pass the result of a previous call to LPI2C_SlaveGetStatusFlags() .

See Also

[_lpi2c_slave_flags](#).

40.5.5.8 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 _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

40.5.5.9 static void LPI2C_SlaveDisableInterrupts (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 disable. See _lpi2c_slave_flags for the set of constants that should be OR'd together to form the bit mask.

40.5.5.10 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.

40.5.5.11 `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.

40.5.5.12 `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.

40.5.5.13 `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.

40.5.5.14 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.

40.5.5.15 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.

40.5.5.16 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.

**40.5.5.17 void LPI2C_SlaveTransferCreateHandle (LPI2C_Type * *base*,
lpi2c_slave_handle_t * *handle*, lpi2c_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.

**40.5.5.18 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 <code>lpi2c_slave_handle_t</code> structure which stores the transfer state.
<i>eventMask</i>	Bit mask formed by OR'ing together lpi2c_slave_transfer_event_t 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 kLPI2C_SlaveAllEvents to enable all events.

Return values

<i>kStatus_Success</i>	Slave transfers were successfully started.
<i>kStatus_LPI2C_Busy</i>	Slave transfers have already been started on this handle.

40.5.5.19 `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 <code>i2c_slave_handle_t</code> 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_NoTransferInProgress</i>	

40.5.5.20 `void LPI2C_SlaveTransferAbort (LPI2C_Type * base, lpi2c_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.

Return values

<i>kStatus_Success</i>	
<i>kStatus_LPI2C_Idle</i>	

40.5.5.21 void LPI2C_SlaveTransferHandleIRQ (LPI2C_Type * *base*, lpi2c_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.

40.6 LPI2C Master DMA Driver

40.6.1 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.

40.6.2 Data Structure Documentation

40.6.2.1 struct [_lpi2c_master_edma_handle](#)

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.

40.6.3 Typedef Documentation

40.6.3.1 `typedef void(* lpi2c_master_edma_transfer_callback_t)(LPI2C_Type *base,
lpi2c_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.

40.6.4 Function Documentation

40.6.4.1 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*)

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_MasterTransferAbortEDMA\(\)](#) 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.

40.6.4.2 status_t LPI2C_MasterTransferEDMA (LPI2C_Type * *base*, lpi2c_master_edma_handle_t * *handle*, lpi2c_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.

40.6.4.3 **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.
out	<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.

40.6.4.4 **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.

40.7 LPI2C FreeRTOS Driver

40.7.1 Overview

Driver version

- #define `FSL_LPI2C_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 0)`)
LPI2C FreeRTOS driver version.

LPI2C RTOS Operation

- `status_t LPI2C_RTOS_Init` (`lpi2c_rtos_handle_t *handle`, `LPI2C_Type *base`, `const lpi2c_master_config_t *masterConfig`, `uint32_t srcClock_Hz`)
Initializes LPI2C.
- `status_t LPI2C_RTOS_Deinit` (`lpi2c_rtos_handle_t *handle`)
Deinitializes the LPI2C.
- `status_t LPI2C_RTOS_Transfer` (`lpi2c_rtos_handle_t *handle`, `lpi2c_master_transfer_t *transfer`)
Performs I2C transfer.

40.7.2 Macro Definition Documentation

40.7.2.1 #define FSL_LPI2C_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

40.7.3 Function Documentation

40.7.3.1 `status_t LPI2C_RTOS_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

<i>handle</i>	The RTOS LPI2C handle, the pointer to an allocated space for RTOS context.
<i>base</i>	The pointer base address of the LPI2C instance to initialize.
<i>masterConfig</i>	Configuration structure to set-up LPI2C in master mode.
<i>srcClock_Hz</i>	Frequency of input clock of the LPI2C module.

Returns

status of the operation.

40.7.3.2 `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.
---------------	------------------------

**40.7.3.3 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.

40.8 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.

40.8.1 LPI2C CMSIS Driver

40.8.1.1 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;
```

40.8.1.2 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;

```

40.8.1.3 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 41

LPSPI: Low Power Serial Peripheral Interface

41.1 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](#)

41.2 LPSPI Peripheral driver

41.2.1 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.

41.2.2 Function groups

41.2.2.1 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.

41.2.2.2 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.

41.2.2.3 LPSPI Transfer Operation

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

41.2.2.4 LPSPI Status Operation

This function group gets/clears the LPSPI status.

41.2.2.5 LPSPI Block Transfer Operation

This function group transfers a block of data, gets the transfer status, and aborts the transfer.

41.2.3 Typical use case

41.2.3.1 Master Operation

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

41.2.3.2 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_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_MBF_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 }
LPSPI DMA source.
- enum `lpspi_master_slave_mode_t` {
`kLPSPI_Master` = 1U,
`kLPSPI_Slave` = 0U }
LPSPI master or slave mode configuration.
- 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_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_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, 2, 1))
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 [LPSPI_GetInstance](#) ([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_SetPCSContinuous](#) (LPSPI_Type *base, bool IsContinuous)
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.

41.2.4 Data Structure Documentation

41.2.4.1 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_data_out_config_t dataOutConfig](#)
Configures if the output data is tristated between accesses (LPSPI_PCS is negated).

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_data_out_config_t lpspi_master_config_t::dataOutConfig`

41.2.4.2 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`

41.2.4.3 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.

41.2.4.4 struct _lpspi_master_handle

Forward declaration of the [_lpspi_master_handle](#) typedefs.

Data Fields

- volatile bool `isPcsContinuous`
Is PCS continuous in transfer.
- volatile bool `writeTcrInIsr`
A flag that whether should write TCR in ISR.
- volatile bool `isByteSwap`
A flag that whether should byte swap.
- volatile bool `isTxMask`
A flag that whether TCR[TXMSK] is set.
- volatile uint16_t `bytesPerFrame`
Number of bytes in each frame.
- 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.
- 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`

41.2.4.5 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`

41.2.5 Macro Definition Documentation

41.2.5.1 #define FSL_LP_SPI_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

41.2.5.2 #define LP_SPI_DUMMY_DATA (0x00U)

Dummy data used for tx if there is not txData.

41.2.5.3 #define SPI_RETRY_TIMES 0U /* Define to zero means keep waiting until the flag is assert/deassert. */

41.2.5.4 #define LP_SPI_MASTER_PCS_SHIFT (4U)

41.2.5.5 #define LP_SPI_MASTER_PCS_MASK (0xF0U)

41.2.5.6 #define LP_SPI_SLAVE_PCS_SHIFT (4U)

41.2.5.7 #define LP_SPI_SLAVE_PCS_MASK (0xF0U)

41.2.6 Typedef Documentation

41.2.6.1 typedef void(* lpspi_master_transfer_callback_t)(LP_SPI_Type *base, lpspi_master_handle_t *handle, status_t status, void *userData)

Parameters

<i>base</i>	LP_SPI peripheral address.
<i>handle</i>	Pointer to the handle for the LP_SPI 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.

41.2.6.2 typedef void(* lpspi_slave_transfer_callback_t)(LP_SPI_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.

41.2.7 Enumeration Type Documentation

41.2.7.1 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.

41.2.7.2 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.

41.2.7.3 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.

41.2.7.4 enum _lpspi_dma_enable

Enumerator

kLPSPI_TxDmaEnable Transmit data DMA enable.
kLPSPI_RxDmaEnable Receive data DMA enable.

41.2.7.5 enum lpspi_master_slave_mode_t

Enumerator

kLPSPI_Master LPSPI peripheral operates in master mode.
kLPSPI_Slave LPSPI peripheral operates in slave mode.

41.2.7.6 enum lpspi_which_pcs_t

Enumerator

kLPSPI_Pcs0 PCS[0].
kLPSPI_Pcs1 PCS[1].
kLPSPI_Pcs2 PCS[2].
kLPSPI_Pcs3 PCS[3].

41.2.7.7 enum lpspi_pcs_polarity_config_t

Enumerator

kLPSPI_PcsActiveHigh PCS Active High (idles low)
kLPSPI_PcsActiveLow PCS Active Low (idles high)

41.2.7.8 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).

41.2.7.9 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)

41.2.7.10 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.

41.2.7.11 enum lpspi_shift_direction_t

Enumerator

kLPSPi_MsbFirst Data transfers start with most significant bit.
kLPSPi_LsbFirst Data transfers start with least significant bit.

41.2.7.12 enum lpspi_host_request_select_t

Enumerator

kLPSPi_HostReqExtPin Host Request is an ext pin.
kLPSPi_HostReqInternalTrigger Host Request is an internal trigger.

41.2.7.13 enum lpspi_match_config_t

Enumerator

kLPSPi_MatchDisabled LPSPi Match Disabled.
kLPSPi_1stWordEqualsM0orM1 LPSPi Match Enabled.
kLPSPi_AnyWordEqualsM0orM1 LPSPi Match Enabled.

kLPSPi_1stWordEqualsM0and2ndWordEqualsM1 LPSPi Match Enabled.
kLPSPi_AnyWordEqualsM0andNxtWordEqualsM1 LPSPi Match Enabled.
kLPSPi_1stWordAndM1EqualsM0andM1 LPSPi Match Enabled.
kLPSPi_AnyWordAndM1EqualsM0andM1 LPSPi Match Enabled.

41.2.7.14 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.

41.2.7.15 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.

41.2.7.16 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]

41.2.7.17 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.

41.2.7.18 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_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.

41.2.7.19 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.

41.2.7.20 enum _lpspi_transfer_state

Enumerator

- kLPSPi_Idle*** Nothing in the transmitter/receiver.

kLPSPi_Busy Transfer queue is not finished.
kLPSPi_Error Transfer error.

41.2.8 Function Documentation

41.2.8.1 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

41.2.8.2 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 lpspi_master_config_t structure
---------------------	--

41.2.8.3 void LPSPi_SlaveInit (LPSPi_Type * *base*, const lpspi_slave_config_t * *slaveConfig*)

Parameters

<i>base</i>	LPSPi peripheral address.
<i>slaveConfig</i>	Pointer to a structure lpspi_slave_config_t .

41.2.8.4 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 lpspi_slave_config_t structure.
--------------------	--

41.2.8.5 void LPSPI_Deinit (LPSPI_Type * *base*)

Call this API to disable the LPSPI clock.

Parameters

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

41.2.8.6 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.
-------------	---------------------------

41.2.8.7 uint32_t LPSPI_GetInstance (LPSPI_Type * *base*)

Parameters

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

Returns

LPSPI instance.

41.2.8.8 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.

**41.2.8.9 static uint32_t LPSPI_GetStatusFlags (LPSPI_Type * *base*) [inline],
[static]**

Parameters

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

Returns

The LPSPI status(in SR register).

**41.2.8.10 static uint8_t LPSPI_GetTxFifoSize (LPSPI_Type * *base*) [inline],
[static]**

Parameters

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

Returns

The LPSPI Tx FIFO size.

**41.2.8.11 static uint8_t LPSPI_GetRxFifoSize (LPSPI_Type * *base*) [inline],
[static]**

Parameters

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

Returns

The LPSPI Rx FIFO size.

**41.2.8.12 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.

41.2.8.13 `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.

41.2.8.14 `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 <code>_lpspi_flags</code> .

< The status flags are cleared by writing 1 (w1c).

41.2.8.15 `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 <code>_lpspi_interrupt_enable</code> .

41.2.8.16 **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 <code>_lpspi_interrupt_enable</code> .

41.2.8.17 **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_RxDmaEnable);
*
```

Parameters

<i>base</i>	LPSPi peripheral address.
<i>mask</i>	The interrupt mask; Use the enum <code>_lpspi_dma_enable</code> .

41.2.8.18 **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_RxDmaEnable);
*
```

Parameters

<i>base</i>	LPSPI peripheral address.
<i>mask</i>	The interrupt mask; Use the enum <code>_lpspi_dma_enable</code> .

41.2.8.19 **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.

41.2.8.20 **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.

41.2.8.21 **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.

**41.2.8.22 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.

**41.2.8.23 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.

**41.2.8.24 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>IsContinous</i>	True to set the transfer PCS to continuous mode, false to set to uncontinuous mode.

41.2.8.25 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.

41.2.8.26 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.

41.2.8.27 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.

41.2.8.28 **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 <code>_lpspi_pcs_polarity</code> .

41.2.8.29 **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.

41.2.8.30 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.

41.2.8.31 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> .

41.2.8.32 `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>delayTimeInNanoSec</i>	The desired delay value in nano-seconds.
<i>whichDelay</i>	The desired delay to configuration, which must be of type <code>lpspi_delay_type_t</code> .
<i>srcClock_Hz</i>	Module source input clock in Hertz.

Returns

actual Calculated delay value in nano-seconds.

41.2.8.33 `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.

41.2.8.34 `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.

41.2.8.35 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.

41.2.8.36 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.

41.2.8.37 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.

41.2.8.38 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.

41.2.8.39 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 <code>lpspi_master_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`.

41.2.8.40 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 <code>lpspi_master_handle_t</code> structure which stores the transfer state.

41.2.8.41 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 <code>lpspi_master_handle_t</code> structure which stores the transfer state.

41.2.8.42 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 <code>lpspi_slave_handle_t</code> .
<i>callback</i>	DSPI callback.
<i>userData</i>	callback function parameter.

41.2.8.43 `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`.

41.2.8.44 `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`.

41.2.8.45 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.

**41.2.8.46 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 <code>lpspi_slave_handle_t</code> structure which stores the transfer state.

41.2.9 Variable Documentation

41.2.9.1 volatile uint8_t g_lpspiDummyData[]

41.3 LPSPI eDMA Driver

41.3.1 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.
- 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, 1, 0)`)
LPSPI EDMA driver version.

41.3.2 Data Structure Documentation

41.3.2.1 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`
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.
- 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`

41.3.2.2 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`

41.3.3 Macro Definition Documentation

41.3.3.1 `#define FSL_LPSPI_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

41.3.4 Typedef Documentation

41.3.4.1 `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.

41.3.4.2 `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.

41.3.5 Function Documentation

**41.3.5.1 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 Rx 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 edmaIntermediaryToTxRegHandle. (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 lpspi_master_edma_handle_t.
<i>callback</i>	LPSPI callback.
<i>userData</i>	callback function parameter.
<i>edmaRxRegTo-RxDataHandle</i>	edmaRxRegToRxDataHandle pointer to edma_handle_t .
<i>edmaTxData-ToTxReg-Handle</i>	edmaTxDataToTxRegHandle pointer to edma_handle_t .

**41.3.5.2 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 lpspi_transfer_t structure.

Returns

status of `status_t`.

41.3.5.3 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 <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state.

41.3.5.4 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 <code>lpspi_master_edma_handle_t</code> structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the EDMA transaction.

Returns

status of `status_t`.

41.3.5.5 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>edmaRxRegToRxDataHandle</i>	edmaRxRegToRxDataHandle pointer to edma_handle_t .
<i>edmaTxDataToTxRegHandle</i>	edmaTxDataToTxRegHandle pointer to edma_handle_t .

41.3.5.6 **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 lpspi_transfer_t structure.

Returns

status of status_t.

**41.3.5.7 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 <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state.

**41.3.5.8 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 <code>lpspi_slave_edma_handle_t</code> structure which stores the transfer state.
<i>count</i>	Number of bytes transferred so far by the eDMA transaction.

Returns

status of `status_t`.

41.4 LPSPI FreeRTOS Driver

41.4.1 Overview

Driver version

- #define `FSL_LPSPI_FREERTOS_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 5)`)
LPSPI FreeRTOS driver version 2.0.5.

LPSPI RTOS Operation

- `status_t LPSPI_RTOS_Init` (`lpspi_rtos_handle_t *handle`, `LPSPI_Type *base`, `const lpspi_master_config_t *masterConfig`, `uint32_t srcClock_Hz`)
Initializes LPSPI.
- `status_t LPSPI_RTOS_Deinit` (`lpspi_rtos_handle_t *handle`)
Deinitializes the LPSPI.
- `status_t LPSPI_RTOS_Transfer` (`lpspi_rtos_handle_t *handle`, `lpspi_transfer_t *transfer`)
Performs SPI transfer.

41.4.2 Macro Definition Documentation

41.4.2.1 #define FSL_LPSPI_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))

41.4.3 Function Documentation

41.4.3.1 `status_t LPSPI_RTOS_Init (lpspi_rtos_handle_t * handle, LPSPI_Type * base, const lpspi_master_config_t * masterConfig, uint32_t srcClock_Hz)`

This function initializes the LPSPI module and related RTOS context.

Parameters

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

Returns

status of the operation.

41.4.3.2 status_t LPSPI_RTOS_Deinit (lpspi_rtos_handle_t * *handle*)

This function deinitializes the LPSPI module and related RTOS context.

Parameters

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

41.4.3.3 status_t LPSPI_RTOS_Transfer (lpspi_rtos_handle_t * *handle*, lpspi_transfer_t * *transfer*)

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

Parameters

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

Returns

status of the operation.

41.5 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>.

41.5.1 Function groups

41.5.1.1 LPSPI CMSIS GetVersion Operation

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

41.5.1.2 LPSPI CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

41.5.1.3 LPSPI CMSIS Initialize and Uninitialize Operation

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

41.5.1.4 LPSPI Transfer Operation

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

41.5.1.5 LPSPI Status Operation

This function group gets the LPSPI transfer status.

41.5.1.6 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.

41.5.2 Typical use case

41.5.2.1 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 uninitialized */
Driver_SPI0.Uninitialize();

```

41.5.2.2 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 uninitialized */
Driver_SPI2.Uninitialize();

```



Chapter 42

LPUART: Low Power Universal Asynchronous Receiver/-Transmitter Driver

42.1 Overview

Modules

- [LPUART CMSIS Driver](#)
- [LPUART Driver](#)
- [LPUART FreeRTOS Driver](#)
- [LPUART eDMA Driver](#)

42.2 LPUART Driver

42.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

42.2.2 Typical use case

42.2.2.1 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_LBKDIE_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, 5, 2)`)
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 uint32_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 [LPUART_GetInstance](#) (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.

42.2.3 Data Structure Documentation

42.2.3.1 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](#)

- *RX FIFO watermark.*
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](#)

42.2.3.2 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](#)

42.2.3.3 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`

42.2.4 Macro Definition Documentation

42.2.4.1 `#define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))`

42.2.4.2 `#define UART_RETRY_TIMES 0U /* Defining to zero means to keep waiting for the flag until it is assert/deassert. */`

42.2.5 Typedef Documentation

42.2.5.1 `typedef void(* lpuart_transfer_callback_t)(LPUART_Type *base, lpuart_handle_t *handle, status_t status, void *userData)`

42.2.6 Enumeration Type Documentation

42.2.6.1 anonymous enum

Enumerator

kStatus_LPUART_TxBusy TX busy.
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_RxRingBufferOverflow LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverflow 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.

42.2.6.2 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.

42.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.

kLPUART_SevenDataBits Seven data bit.

42.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.

kLPUART_TwoStopBit Two stop bits.

42.2.6.5 enum lpuart_transmit_cts_source_t

Enumerator

kLPUART_CtsSourcePin CTS resource is the LPUART_CTS pin.

kLPUART_CtsSourceMatchResult CTS resource is the match result.

42.2.6.6 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.

42.2.6.7 enum lpuart_idle_type_select_t

Enumerator

kLPUART_IdleTypeStartBit Start counting after a valid start bit.
kLPUART_IdleTypeStopBit Start counting after a stop bit.

42.2.6.8 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.

42.2.6.9 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

42.2.6.10 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

42.2.7 Function Documentation

42.2.7.1 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.
-------------	---------------------------------

42.2.7.2 **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_GetDefault-Config\(\)](#) function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the 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

42.2.7.3 **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.
-------------	---------------------------------

42.2.7.4 void LPUART_GetDefaultConfig (lpuart_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.
---------------	---------------------------------------

42.2.7.5 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, 200000000U);
*
```

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.

42.2.7.6 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.

42.2.7.7 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.

42.2.7.8 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.

42.2.7.9 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.

42.2.7.10 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.

42.2.7.11 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`.

42.2.7.12 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_NoiseErrorInRxDataRegFlag`, `kLPUART_ParityErrorInRxDataRegFlag`, `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.

42.2.7.13 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 the enumeration <code>_uart_interrupt_enable</code> .

42.2.7.14 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 .

42.2.7.15 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](#).

**42.2.7.16 static uint32_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.

**42.2.7.17 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.

42.2.7.18 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.

42.2.7.19 uint32_t LPUART_GetInstance (LPUART_Type * *base*)

Parameters

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

Returns

LPUART instance.

42.2.7.20 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.

42.2.7.21 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.

42.2.7.22 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.

42.2.7.23 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.

42.2.7.24 static uint8_t LPUART_GetRxFifoCount (LPUART_Type * *base*) [inline], [static]

Parameters

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

Returns

rx FIFO data count.

42.2.7.25 `static uint8_t LPUART_GetTxFifoCount (LPUART_Type * base) [inline],
[static]`

Parameters

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

Returns

tx FIFO data count.

42.2.7.26 `void LPUART_SendAddress (LPUART_Type * base, uint8_t address)`

Parameters

<i>base</i>	LPUART peripheral base address.
<i>address</i>	LPUART slave address.

42.2.7.27 `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 data 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.

42.2.7.28 **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.

42.2.7.29 **void LPUART_TransferCreateHandle (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_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.

42.2.7.30 **status_t LPUART_TransferSendNonBlocking (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_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 lpuart_transfer_t .

Return values

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

42.2.7.31 **void LPUART_TransferStartRingBuffer (LPUART_Type * *base*, lpuart_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.

42.2.7.32 void LPUART_TransferStopRingBuffer (LPUART_Type * *base*, lpuart_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.

42.2.7.33 size_t LPUART_TransferGetRxRingBufferLength (LPUART_Type * *base*, lpuart_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.

42.2.7.34 void LPUART_TransferAbortSend (LPUART_Type * *base*, lpuart_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.

42.2.7.35 **status_t LPUART_TransferGetSendCount (LPUART_Type * *base*, lpuart_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> ;

42.2.7.36 **status_t LPUART_TransferReceiveNonBlocking (LPUART_Type * *base*, lpuart_handle_t * *handle*, lpuart_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.

42.2.7.37 void LPUART_TransferAbortReceive (LPUART_Type * *base*, lpuart_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.

42.2.7.38 status_t LPUART_TransferGetReceiveCount (LPUART_Type * *base*, lpuart_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 count;

42.2.7.39 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.

42.2.7.40 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.

42.3 LPUART eDMA Driver

42.3.1 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, 5, 2)`)
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.

42.3.2 Data Structure Documentation

42.3.2.1 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](#)

42.3.3 Macro Definition Documentation

42.3.3.1 #define FSL_LPUART_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 5, 2))

42.3.4 Typedef Documentation

42.3.4.1 `typedef void(* lpuart_edma_transfer_callback_t)(LPUART_Type *base,
lpuart_edma_handle_t *handle, status_t status, void *userData)`

42.3.5 Function Documentation

42.3.5.1 `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.

42.3.5.2 `status_t LPUART_SendEDMA (LPUART_Type * base, lpuart_edma_handle_t *
handle, lpuart_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 lpuart_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.

42.3.5.3 **status_t LPUART_ReceiveEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, lpuart_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.

42.3.5.4 **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.

42.3.5.5 **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 <code>lpuart_edma_handle_t</code> structure.

42.3.5.6 `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_NoTransferInProgress</i>	No send in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter <code>count</code> ;

42.3.5.7 `status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type * base, lpuart_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_NoTransferInProgress</i>	No receive in progress.
<i>kStatus_InvalidArgument</i>	Parameter is invalid.
<i>kStatus_Success</i>	Get successfully through the parameter count;

42.3.5.8 void LPUART_TransferEdmaHandleIRQ (LPUART_Type * *base*, void * *lpuartEdmaHandle*)

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.

42.4 LPUART FreeRTOS Driver

42.4.1 Overview

Data Structures

- struct [lpuart_rtos_config_t](#)
LPUART RTOS configuration structure. [More...](#)

Driver version

- #define [FSL_LPUART_FREERTOS_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 5, 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.

42.4.2 Data Structure Documentation

42.4.2.1 struct lpuart_rtos_config_t

Data Fields

- LPUART_Type * [base](#)
UART base address.
- uint32_t [srcclk](#)
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.
- `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.

42.4.3 Macro Definition Documentation

42.4.3.1 `#define FSL_LPUART_FREERTOS_DRIVER_VERSION (MAKE_VERSION(2, 5, 0))`

42.4.4 Function Documentation

42.4.4.1 `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

42.4.4.2 `int LPUART_RTOS_Deinit (lpuart_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.
---------------	-------------------------

42.4.4.3 int LPUART_RTOS_Send (lpuart_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.

42.4.4.4 int LPUART_RTOS_Receive (lpuart_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.

42.5 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.

42.5.1 Function groups

42.5.1.1 LPUART CMSIS GetVersion Operation

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

42.5.1.2 LPUART CMSIS GetCapabilities Operation

This function group will return the capabilities of this driver.

42.5.1.3 LPUART CMSIS Initialize and Uninitialize Operation

This function will initialize and uninitialize 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 slected 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.

42.5.1.4 LPUART CMSIS Transfer Operation

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

42.5.1.5 LPUART CMSIS Status Operation

This function group gets the LPUART transfer status.

42.5.1.6 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 43

MECC: internal error correction code

43.1 Overview

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

The MECC64 module supports Single Error Correction and Double Error Detection(SECDED) ECC function to provide reliability for 4 banks On-Chip RAM(OCRAM) access.

This example code shows how to correct single error and detect multiple error using the MECC driver.

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

Data Structures

- struct `mecc_config_t`
MECC user configuration. [More...](#)
- struct `mecc_single_error_info_t`
MECC ocram single error information, including single error address, ECC code, error data and error bit position. [More...](#)
- struct `mecc_multi_error_info_t`
MECC ocram multiple error information, including multiple error address, ECC code, error data. [More...](#)

Enumerations

- enum { `kStatus_MECC_BankMiss` = MAKE_STATUS(kStatusGroup_MECC, 0) }
Error codes for the MECC driver.
- enum {
 `kMECC_SingleError0InterruptEnable`,
 `kMECC_SingleError1InterruptEnable`,
 `kMECC_SingleError2InterruptEnable`,
 `kMECC_SingleError3InterruptEnable`,
 `kMECC_MultiError0InterruptEnable`,
 `kMECC_MultiError1InterruptEnable`,
 `kMECC_MultiError2InterruptEnable`,
 `kMECC_MultiError3InterruptEnable`,
 `kMECC_StrobeError0InterruptEnable`,
 `kMECC_StrobeError1InterruptEnable`,
 `kMECC_StrobeError2InterruptEnable`,
 `kMECC_StrobeError3InterruptEnable`,
 `kMECC_AccessError0InterruptEnable`,
 `kMECC_AccessError1InterruptEnable`,
 `kMECC_AccessError2InterruptEnable`,
 `kMECC_AccessError3InterruptEnable`,
 `kMECC_AllInterruptsEnable` = 0xFFFF }
}

MECC interrupt configuration structure, default settings all disabled.

- enum {
 - kMECC_SingleError0InterruptStatusEnable,
 - kMECC_SingleError1InterruptStatusEnable,
 - kMECC_SingleError2InterruptStatusEnable,
 - kMECC_SingleError3InterruptStatusEnable,
 - kMECC_MultiError0InterruptStatusEnable,
 - kMECC_MultiError1InterruptStatusEnable,
 - kMECC_MultiError2InterruptStatusEnable,
 - kMECC_MultiError3InterruptStatusEnable,
 - kMECC_StrobeError0InterruptStatusEnable,
 - kMECC_StrobeError1InterruptStatusEnable,
 - kMECC_StrobeError2InterruptStatusEnable,
 - kMECC_StrobeError3InterruptStatusEnable,
 - kMECC_AccessError0InterruptStatusEnable,
 - kMECC_AccessError1InterruptStatusEnable,
 - kMECC_AccessError2InterruptStatusEnable,
 - kMECC_AccessError3InterruptStatusEnable,
 - kMECC_AllInterruptsStatusEnable = 0xFFFF }

MECC interrupt status configuration structure, default settings all disabled.

- enum {
 - kMECC_SingleError0InterruptFlag,
 - kMECC_SingleError1InterruptFlag,
 - kMECC_SingleError2InterruptFlag,
 - kMECC_SingleError3InterruptFlag,
 - kMECC_MultiError0InterruptFlag,
 - kMECC_MultiError1InterruptFlag,
 - kMECC_MultiError2InterruptFlag,
 - kMECC_MultiError3InterruptFlag,
 - kMECC_StrobeError0InterruptFlag,
 - kMECC_StrobeError1InterruptFlag,
 - kMECC_StrobeError2InterruptFlag,
 - kMECC_StrobeError3InterruptFlag,
 - kMECC_AccessError0InterruptFlag = MECC_ERR_STATUS_ADDR_ERR0_MASK,
 - kMECC_AccessError1InterruptFlag = MECC_ERR_STATUS_ADDR_ERR1_MASK,
 - kMECC_AccessError2InterruptFlag = MECC_ERR_STATUS_ADDR_ERR2_MASK,
 - kMECC_AccessError3InterruptFlag = MECC_ERR_STATUS_ADDR_ERR3_MASK,
 - kMECC_AllInterruptsFlag = 0xFFFF }

MECC status flags.

- enum {
 - kMECC_OcramBank0 = 0U,
 - kMECC_OcramBank1 = 1U,
 - kMECC_OcramBank2 = 2U,
 - kMECC_OcramBank3 = 3U }

MECC ocram bank number.

- enum {


```
kMECC_Instance0 = 0U,
kMECC_Instance1 = 1U }
    MECC instance.
```

Driver version

- #define `FSL_MECC_DRIVER_VERSION` (`MAKE_VERSION(2U, 0U, 2U)`)
Driver version 2.0.2.

Initialization and deinitialization

- void `MECC_Init` (`MECC_Type *base`, `mecc_config_t *config`)
MECC module initialization function.
- void `MECC_Deinit` (`MECC_Type *base`)
Deinitializes the MECC.
- void `MECC_GetDefaultConfig` (`mecc_config_t *config`)
Sets the MECC configuration structure to default values.

Status

- static uint32_t `MECC_GetStatusFlags` (`MECC_Type *base`)
Gets MECC status flags.
- static void `MECC_ClearStatusFlags` (`MECC_Type *base`, uint32_t mask)
MECC module clear interrupt status.
- static void `MECC_EnableInterruptStatus` (`MECC_Type *base`, uint32_t mask)
MECC module enable interrupt status.
- static void `MECC_DisableInterruptStatus` (`MECC_Type *base`, uint32_t mask)
MECC module disable interrupt status.

Interrupts

- static void `MECC_EnableInterrupts` (`MECC_Type *base`, uint32_t mask)
MECC module enable interrupt.
- static void `MECC_DisableInterrupts` (`MECC_Type *base`, uint32_t mask)
MECC module disable interrupt.

functional

- `status_t MECC_ErrorInjection` (`MECC_Type *base`, uint32_t lowererrordata, uint32_t higherrordata, uint8_t eccdata, uint8_t banknumber)
MECC module error injection.
- `status_t MECC_GetSingleErrorInfo` (`MECC_Type *base`, `mecc_single_error_info_t *info`, uint8_t banknumber)
MECC module get single error information.
- `status_t MECC_GetMultiErrorInfo` (`MECC_Type *base`, `mecc_multi_error_info_t *info`, uint8_t banknumber)
MECC module get multiple error information.

43.2 Data Structure Documentation

43.2.1 struct mecc_config_t

Data Fields

- bool [enableMecc](#)
Enable the MECC function.
- uint32_t [Ocram1StartAddress](#)
Ocram 1 start address.
- uint32_t [Ocram1EndAddress](#)
Ocram 1 end address.
- uint32_t [Ocram2StartAddress](#)
Ocram 2 start address.
- uint32_t [Ocram2EndAddress](#)
Ocram 2 end address.

Field Documentation

- (1) bool mecc_config_t::enableMecc
- (2) uint32_t mecc_config_t::Ocram1StartAddress
- (3) uint32_t mecc_config_t::Ocram1EndAddress
- (4) uint32_t mecc_config_t::Ocram2StartAddress
- (5) uint32_t mecc_config_t::Ocram2EndAddress

43.2.2 struct mecc_single_error_info_t

Data Fields

- uint32_t [singleErrorAddress](#)
Single error address on Ocram bank n.
- uint32_t [singleErrorDataLow](#)
Single error low 32 bits uncorrected read data on Ocram bank n.
- uint32_t [singleErrorDataHigh](#)
Single error high 32 bits uncorrected read data on Ocram bank n.
- uint32_t [singleErrorPosLow](#)
Single error bit position of low 32 bits read data on Ocram bank n.
- uint32_t [singleErrorPosHigh](#)
Single error bit position of high 32 bits read data on Ocram bank n.
- uint8_t [singleErrorEccCode](#)
Single error ECC code on Ocram bank n.

43.2.3 struct mecc_multi_error_info_t

Data Fields

- uint32_t [multiErrorAddress](#)
Multiple error address on Ocram bank *n*.
- uint32_t [multiErrorDataLow](#)
Multiple error low 32 bits read data on Ocram bank *n*.
- uint32_t [multiErrorDataHigh](#)
Multiple error high 32 bits read data on Ocram bank *n*.
- uint8_t [multiErrorEccCode](#)
Multiple error ECC code on Ocram bank *n*.

43.3 Macro Definition Documentation

43.3.1 #define FSL_MECC_DRIVER_VERSION (MAKE_VERSION(2U, 0U, 2U))

43.4 Enumeration Type Documentation

43.4.1 anonymous enum

Enumerator

kStatus_MECC_BankMiss Ocram bank miss.

43.4.2 anonymous enum

This structure contains the settings for all of the MECC interrupt configurations.

Enumerator

<i>kMECC_SingleError0InterruptEnable</i>	Single Bit Error On Ocram Bank0 interrupt enable.
<i>kMECC_SingleError1InterruptEnable</i>	Single Bit Error On Ocram Bank1 interrupt enable.
<i>kMECC_SingleError2InterruptEnable</i>	Single Bit Error On Ocram Bank2 interrupt enable.
<i>kMECC_SingleError3InterruptEnable</i>	Single Bit Error On Ocram Bank3 interrupt enable.
<i>kMECC_MultiError0InterruptEnable</i>	Multiple Bits Error On Ocram Bank0 interrupt enable.
<i>kMECC_MultiError1InterruptEnable</i>	Multiple Bits Error On Ocram Bank1 interrupt enable.
<i>kMECC_MultiError2InterruptEnable</i>	Multiple Bits Error On Ocram Bank2 interrupt enable.
<i>kMECC_MultiError3InterruptEnable</i>	Multiple Bits Error On Ocram Bank3 interrupt enable.
<i>kMECC_StrobeError0InterruptEnable</i>	AXI Strobe Error On Ocram Bank0 interrupt enable.
<i>kMECC_StrobeError1InterruptEnable</i>	AXI Strobe Error On Ocram Bank1 interrupt enable.
<i>kMECC_StrobeError2InterruptEnable</i>	AXI Strobe Error On Ocram Bank2 interrupt enable.
<i>kMECC_StrobeError3InterruptEnable</i>	AXI Strobe Error On Ocram Bank3 interrupt enable.
<i>kMECC_AccessError0InterruptEnable</i>	Ocram Access Error On Bank0 interrupt enable.
<i>kMECC_AccessError1InterruptEnable</i>	Ocram Access Error On Bank1 interrupt enable.
<i>kMECC_AccessError2InterruptEnable</i>	Ocram Access Error On Bank2 interrupt enable.

kMECC_AccessError3InterruptEnable Ocram Access Error On Bank3 interrupt enable.

kMECC_AllInterruptsEnable all interrupts enable

43.4.3 anonymous enum

This structure contains the settings for all of the MECC interrupt status configurations.

Enumerator

<i>kMECC_SingleError0InterruptStatusEnable</i>	Single Bit Error On Ocram Bank0 interrupt status enable.
<i>kMECC_SingleError1InterruptStatusEnable</i>	Single Bit Error On Ocram Bank1 interrupt status enable.
<i>kMECC_SingleError2InterruptStatusEnable</i>	Single Bit Error On Ocram Bank2 interrupt status enable.
<i>kMECC_SingleError3InterruptStatusEnable</i>	Single Bit Error On Ocram Bank3 interrupt status enable.
<i>kMECC_MultiError0InterruptStatusEnable</i>	Multiple Bits Error On Ocram Bank0 interrupt status enable.
<i>kMECC_MultiError1InterruptStatusEnable</i>	Multiple Bits Error On Ocram Bank1 interrupt status enable.
<i>kMECC_MultiError2InterruptStatusEnable</i>	Multiple Bits Error On Ocram Bank2 interrupt status enable.
<i>kMECC_MultiError3InterruptStatusEnable</i>	Multiple Bits Error On Ocram Bank3 interrupt status enable.
<i>kMECC_StrobeError0InterruptStatusEnable</i>	AXI Strobe Error On Ocram Bank0 interrupt status enable.
<i>kMECC_StrobeError1InterruptStatusEnable</i>	AXI Strobe Error On Ocram Bank1 interrupt status enable.
<i>kMECC_StrobeError2InterruptStatusEnable</i>	AXI Strobe Error On Ocram Bank2 interrupt status enable.
<i>kMECC_StrobeError3InterruptStatusEnable</i>	AXI Strobe Error On Ocram Bank3 interrupt status enable.
<i>kMECC_AccessError0InterruptStatusEnable</i>	Ocram Access Error On Bank0 interrupt status enable.
<i>kMECC_AccessError1InterruptStatusEnable</i>	Ocram Access Error On Bank1 interrupt status enable.
<i>kMECC_AccessError2InterruptStatusEnable</i>	Ocram Access Error On Bank2 interrupt status enable.
<i>kMECC_AccessError3InterruptStatusEnable</i>	Ocram Access Error On Bank3 interrupt status enable.
<i>kMECC_AllInterruptsStatusEnable</i>	all interrupts enable

43.4.4 anonymous enum

This provides constants for the MECC status flags for use in the MECC functions.

Enumerator

<i>kMECC_SingleError0InterruptFlag</i>	Single Bit Error On Ocram Bank0 interrupt flag.
<i>kMECC_SingleError1InterruptFlag</i>	Single Bit Error On Ocram Bank1 interrupt flag.
<i>kMECC_SingleError2InterruptFlag</i>	Single Bit Error On Ocram Bank2 interrupt flag.
<i>kMECC_SingleError3InterruptFlag</i>	Single Bit Error On Ocram Bank3 interrupt flag.
<i>kMECC_MultiError0InterruptFlag</i>	Multiple Bits Error On Ocram Bank0 interrupt flag.
<i>kMECC_MultiError1InterruptFlag</i>	Multiple Bits Error On Ocram Bank1 interrupt flag.
<i>kMECC_MultiError2InterruptFlag</i>	Multiple Bits Error On Ocram Bank2 interrupt flag.
<i>kMECC_MultiError3InterruptFlag</i>	Multiple Bits Error On Ocram Bank3 interrupt flag.
<i>kMECC_StrobeError0InterruptFlag</i>	AXI Strobe Error On Ocram Bank0 interrupt flag.
<i>kMECC_StrobeError1InterruptFlag</i>	AXI Strobe Error On Ocram Bank1 interrupt flag.
<i>kMECC_StrobeError2InterruptFlag</i>	AXI Strobe Error On Ocram Bank2 interrupt flag.
<i>kMECC_StrobeError3InterruptFlag</i>	AXI Strobe Error On Ocram Bank3 interrupt flag.
<i>kMECC_AccessError0InterruptFlag</i>	Ocram Access Error On Bank0 interrupt flag.
<i>kMECC_AccessError1InterruptFlag</i>	Ocram Access Error On Bank1 interrupt flag.
<i>kMECC_AccessError2InterruptFlag</i>	Ocram Access Error On Bank2 interrupt flag.
<i>kMECC_AccessError3InterruptFlag</i>	Ocram Access Error On Bank3 interrupt flag.
<i>kMECC_AllInterruptsFlag</i>	all interrupts interrupt flag

43.4.5 anonymous enum

Enumerator

<i>kMECC_OcramBank0</i>	ocram bank number 0: ocram_base_address+0x20*i
<i>kMECC_OcramBank1</i>	ocram bank number 1: ocram_base_address+0x20*i+0x8
<i>kMECC_OcramBank2</i>	ocram bank number 2: ocram_base_address+0x20*i+0x10
<i>kMECC_OcramBank3</i>	ocram bank number 3: ocram_base_address+0x20*i+0x18

43.4.6 anonymous enum

Enumerator

<i>kMECC_Instance0</i>	Peripheral MECC1 base.
<i>kMECC_Instance1</i>	Peripheral MECC2 base.

43.5 Function Documentation

43.5.1 void MECC_Init (MECC_Type * *base*, mecc_config_t * *config*)

Parameters

<i>base</i>	MECC base address.
<i>config</i>	pointer to the MECC configuration structure.

43.5.2 void MECC_Deinit (MECC_Type * *base*)

Parameters

<i>base</i>	MECC base address.
-------------	--------------------

43.5.3 void MECC_GetDefaultConfig (mecc_config_t * *config*)

Parameters

<i>config</i>	pointer to the MECC configuration structure.
---------------	--

43.5.4 static uint32_t MECC_GetStatusFlags (MECC_Type * *base*) [inline], [static]

Parameters

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

Returns

MECC status flags.

43.5.5 static void MECC_ClearStatusFlags (MECC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	MECC base address.
<i>mask</i>	status to clear.

43.5.6 static void MECC_EnableInterruptStatus (MECC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	MECC base address.
<i>mask</i>	status to enable.

43.5.7 static void MECC_DisableInterruptStatus (MECC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	MECC base address.
<i>mask</i>	status to disable.

43.5.8 static void MECC_EnableInterrupts (MECC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	MECC base address.
<i>mask</i>	The interrupts to enable.

43.5.9 static void MECC_DisableInterrupts (MECC_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	MECC base address.
<i>mask</i>	The interrupts to disable.

43.5.10 **status_t MECC_ErrorInjection (MECC_Type * *base*, uint32_t *lowererrordata*, uint32_t *higherrordata*, uint8_t *eccdata*, uint8_t *banknumber*)**

Parameters

<i>base</i>	MECC base address.
<i>lowererrordata</i>	low 32 bits data.
<i>higherrordata</i>	high 32 bits data.
<i>eccdata</i>	ecc code.
<i>banknumber</i>	ocram bank number.

Return values

<i>kStatus_Success.</i>	Bank0: ocram_base_address+0x20*i Bank1: ocram_base_address+0x20*i+0x8 Bank2: ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i = 0,1,2,3,4.....
-------------------------	--

43.5.11 **status_t MECC_GetSingleErrorInfo (MECC_Type * *base*, mecc_single_error_info_t * *info*, uint8_t *banknumber*)**

Parameters

<i>base</i>	MECC base address.
<i>info</i>	single error information.
<i>banknumber</i>	ocram bank number.

Return values

<i>kStatus_Success.</i>	
<i>kStatus_MECC_Bank-Miss.</i>	Bank0: ocram_base_address+0x20*i Bank1: ocram_base_- address+0x20*i+0x8 Bank2: ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i = 0,1,2,3,4.....

43.5.12 **status_t MECC_GetMultiErrorInfo (MECC_Type * *base*, mecc_multi_error_info_t * *info*, uint8_t *banknumber*)**

Parameters

<i>base</i>	MECC base address.
<i>info</i>	multiple error information.
<i>banknumber</i>	ocram bank number.

Return values

<i>kStatus_Success.</i>	
<i>kStatus_MECC_Bank-Miss.</i>	Bank0: ocram_base_address+0x20*i Bank1: ocram_base_- address+0x20*i+0x8 Bank2: ocram_base_address+0x20*i+0x10 Bank3: ocram_base_address+0x20*i+0x18 i = 0,1,2,3,4.....

Chapter 44

MIPI CSI2 RX: MIPI CSI2 RX Driver

44.1 Overview

The MCUXpresso SDK provides a peripheral driver for the MIPI CSI-2 RX.

Data Structures

- struct `csi2rx_config_t`
CSI2RX configuration. [More...](#)

Enumerations

- enum `_csi2rx_data_lane` {
 `kCSI2RX_DataLane0` = (1U << 0U),
 `kCSI2RX_DataLane1` = (1U << 1U),
 `kCSI2RX_DataLane2` = (1U << 2U),
 `kCSI2RX_DataLane3` = (1U << 3U) }
 CSI2RX data lanes.
- enum `_csi2rx_payload` {

- kCSI2RX_PayloadGroup0Null = (1U << 0U),
- kCSI2RX_PayloadGroup0Blank = (1U << 1U),
- kCSI2RX_PayloadGroup0Embedded = (1U << 2U),
- kCSI2RX_PayloadGroup0YUV420_8Bit = (1U << 10U),
- kCSI2RX_PayloadGroup0YUV422_8Bit = (1U << 14U),
- kCSI2RX_PayloadGroup0YUV422_10Bit = (1U << 15U),
- kCSI2RX_PayloadGroup0RGB444 = (1U << 16U),
- kCSI2RX_PayloadGroup0RGB555 = (1U << 17U),
- kCSI2RX_PayloadGroup0RGB565 = (1U << 18U),
- kCSI2RX_PayloadGroup0RGB666 = (1U << 19U),
- kCSI2RX_PayloadGroup0RGB888 = (1U << 20U),
- kCSI2RX_PayloadGroup0Raw6 = (1U << 24U),
- kCSI2RX_PayloadGroup0Raw7 = (1U << 25U),
- kCSI2RX_PayloadGroup0Raw8 = (1U << 26U),
- kCSI2RX_PayloadGroup0Raw10 = (1U << 27U),
- kCSI2RX_PayloadGroup0Raw12 = (1U << 28U),
- kCSI2RX_PayloadGroup0Raw14 = (1U << 29U),
- kCSI2RX_PayloadGroup1UserDefined1 = (1U << 0U),
- kCSI2RX_PayloadGroup1UserDefined2 = (1U << 1U),
- kCSI2RX_PayloadGroup1UserDefined3 = (1U << 2U),
- kCSI2RX_PayloadGroup1UserDefined4 = (1U << 3U),
- kCSI2RX_PayloadGroup1UserDefined5 = (1U << 4U),
- kCSI2RX_PayloadGroup1UserDefined6 = (1U << 5U),
- kCSI2RX_PayloadGroup1UserDefined7 = (1U << 6U),
- kCSI2RX_PayloadGroup1UserDefined8 = (1U << 7U) }
- CSI2RX payload type.*
- enum _csi2rx_bit_error {
 - kCSI2RX_BitErrorEccTwoBit = (1U << 0U),
 - kCSI2RX_BitErrorEccOneBit = (1U << 1U) }
- MIPI CSI2RX bit errors.*
- enum csi2rx_ppi_error_t {
 - kCSI2RX_PpiErrorSotHs,
 - kCSI2RX_PpiErrorSotSyncHs,
 - kCSI2RX_PpiErrorEsc,
 - kCSI2RX_PpiErrorSyncEsc,
 - kCSI2RX_PpiErrorControl }
- MIPI CSI2RX PPI error types.*
- enum _csi2rx_interrupt
- MIPI CSI2RX interrupt.*
- enum _csi2rx_ulps_status {

```

kCSI2RX_ClockLaneUlps = (1U << 0U),
kCSI2RX_DataLane0Ulps = (1U << 1U),
kCSI2RX_DataLane1Ulps = (1U << 2U),
kCSI2RX_DataLane2Ulps = (1U << 3U),
kCSI2RX_DataLane3Ulps = (1U << 4U),
kCSI2RX_ClockLaneMark = (1U << 5U),
kCSI2RX_DataLane0Mark = (1U << 6U),
kCSI2RX_DataLane1Mark = (1U << 7U),
kCSI2RX_DataLane2Mark = (1U << 8U),
kCSI2RX_DataLane3Mark = (1U << 9U) }

```

MIPI CSI2RX D-PHY ULPS state.

Functions

- void [CSI2RX_Init](#) (MIPI_CSI2RX_Type *base, const [csi2rx_config_t](#) *config)
Enables and configures the CSI2RX peripheral module.
- void [CSI2RX_Deinit](#) (MIPI_CSI2RX_Type *base)
Disables the CSI2RX peripheral module.
- static uint32_t [CSI2RX_GetBitError](#) (MIPI_CSI2RX_Type *base)
Gets the MIPI CSI2RX bit error status.
- static uint32_t [CSI2RX_GetEccBitErrorPosition](#) (uint32_t bitError)
Get ECC one bit error bit position.
- static uint32_t [CSI2RX_GetUlpsStatus](#) (MIPI_CSI2RX_Type *base)
Gets the MIPI CSI2RX D-PHY ULPS status.
- static uint32_t [CSI2RX_GetPpiErrorDataLanes](#) (MIPI_CSI2RX_Type *base, [csi2rx_ppi_error_t](#) errorType)
Gets the MIPI CSI2RX D-PHY PPI error lanes.
- static void [CSI2RX_EnableInterrupts](#) (MIPI_CSI2RX_Type *base, uint32_t mask)
Enable the MIPI CSI2RX interrupts.
- static void [CSI2RX_DisableInterrupts](#) (MIPI_CSI2RX_Type *base, uint32_t mask)
Disable the MIPI CSI2RX interrupts.
- static uint32_t [CSI2RX_GetInterruptStatus](#) (MIPI_CSI2RX_Type *base)
Get the MIPI CSI2RX interrupt status.

Driver version

- #define [FSL_CSI2RX_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 0, 4))
CSI2RX driver version.

44.2 Data Structure Documentation

44.2.1 struct csi2rx_config_t

Data Fields

- uint8_t [laneNum](#)
Number of active lanes used for receiving data.
- uint8_t [tHsSettle_EscClk](#)
Number of rx_clk_esc clock periods for T_HS_SETTLE.

Field Documentation

(1) `uint8_t csi2rx_config_t::laneNum`

(2) `uint8_t csi2rx_config_t::tHsSettle_EscClk`

The T_HS_SETTLE should be in the range of 85ns + 6UI to 145ns + 10UI.

44.3 Macro Definition Documentation

44.3.1 `#define FSL_CSI2RX_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))`

44.4 Enumeration Type Documentation

44.4.1 `enum _csi2rx_data_lane`

Enumerator

kCSI2RX_DataLane0 Data lane 0.

kCSI2RX_DataLane1 Data lane 1.

kCSI2RX_DataLane2 Data lane 2.

kCSI2RX_DataLane3 Data lane 3.

44.4.2 `enum _csi2rx_payload`

Enumerator

kCSI2RX_PayloadGroup0Null NULL.

kCSI2RX_PayloadGroup0Blank Blank.

kCSI2RX_PayloadGroup0Embedded Embedded.

kCSI2RX_PayloadGroup0YUV420_8Bit Legacy YUV420 8 bit.

kCSI2RX_PayloadGroup0YUV422_8Bit YUV422 8 bit.

kCSI2RX_PayloadGroup0YUV422_10Bit YUV422 10 bit.

kCSI2RX_PayloadGroup0RGB444 RGB444.

kCSI2RX_PayloadGroup0RGB555 RGB555.

kCSI2RX_PayloadGroup0RGB565 RGB565.

kCSI2RX_PayloadGroup0RGB666 RGB666.

kCSI2RX_PayloadGroup0RGB888 RGB888.

kCSI2RX_PayloadGroup0Raw6 Raw 6.

kCSI2RX_PayloadGroup0Raw7 Raw 7.

kCSI2RX_PayloadGroup0Raw8 Raw 8.

kCSI2RX_PayloadGroup0Raw10 Raw 10.

kCSI2RX_PayloadGroup0Raw12 Raw 12.

kCSI2RX_PayloadGroup0Raw14 Raw 14.

kCSI2RX_PayloadGroup1UserDefined1 User defined 8-bit data type 1, 0x30.

<i>kCSI2RX_PayloadGroup1UserDefined2</i>	User defined 8-bit data type 2, 0x31.
<i>kCSI2RX_PayloadGroup1UserDefined3</i>	User defined 8-bit data type 3, 0x32.
<i>kCSI2RX_PayloadGroup1UserDefined4</i>	User defined 8-bit data type 4, 0x33.
<i>kCSI2RX_PayloadGroup1UserDefined5</i>	User defined 8-bit data type 5, 0x34.
<i>kCSI2RX_PayloadGroup1UserDefined6</i>	User defined 8-bit data type 6, 0x35.
<i>kCSI2RX_PayloadGroup1UserDefined7</i>	User defined 8-bit data type 7, 0x36.
<i>kCSI2RX_PayloadGroup1UserDefined8</i>	User defined 8-bit data type 8, 0x37.

44.4.3 enum _csi2rx_bit_error

Enumerator

<i>kCSI2RX_BitErrorEccTwoBit</i>	ECC two bit error has occurred.
<i>kCSI2RX_BitErrorEccOneBit</i>	ECC one bit error has occurred.

44.4.4 enum csi2rx_ppi_error_t

Enumerator

<i>kCSI2RX_PpiErrorSotHs</i>	CSI2RX DPHY PPI error ErrSotHS.
<i>kCSI2RX_PpiErrorSotSyncHs</i>	CSI2RX DPHY PPI error ErrSotSync_HS.
<i>kCSI2RX_PpiErrorEsc</i>	CSI2RX DPHY PPI error ErrEsc.
<i>kCSI2RX_PpiErrorSyncEsc</i>	CSI2RX DPHY PPI error ErrSyncEsc.
<i>kCSI2RX_PpiErrorControl</i>	CSI2RX DPHY PPI error ErrControl.

44.4.5 enum _csi2rx_interrupt

44.4.6 enum _csi2rx_ulps_status

Enumerator

<i>kCSI2RX_ClockLaneUlps</i>	Clock lane is in ULPS state.
<i>kCSI2RX_DataLane0Ulps</i>	Data lane 0 is in ULPS state.
<i>kCSI2RX_DataLane1Ulps</i>	Data lane 1 is in ULPS state.
<i>kCSI2RX_DataLane2Ulps</i>	Data lane 2 is in ULPS state.
<i>kCSI2RX_DataLane3Ulps</i>	Data lane 3 is in ULPS state.
<i>kCSI2RX_ClockLaneMark</i>	Clock lane is in mark state.
<i>kCSI2RX_DataLane0Mark</i>	Data lane 0 is in mark state.
<i>kCSI2RX_DataLane1Mark</i>	Data lane 1 is in mark state.
<i>kCSI2RX_DataLane2Mark</i>	Data lane 2 is in mark state.
<i>kCSI2RX_DataLane3Mark</i>	Data lane 3 is in mark state.

44.5 Function Documentation

44.5.1 void CSI2RX_Init (MIPI_CSI2RX_Type * *base*, const csi2rx_config_t * *config*)

Parameters

<i>base</i>	CSI2RX peripheral address.
<i>config</i>	CSI2RX module configuration structure.

44.5.2 void CSI2RX_Deinit (MIPI_CSI2RX_Type * *base*)

Parameters

<i>base</i>	CSI2RX peripheral address.
-------------	----------------------------

44.5.3 static uint32_t CSI2RX_GetBitError (MIPI_CSI2RX_Type * *base*) [inline], [static]

This function gets the RX bit error status, the return value could be compared with [_csi2rx_bit_error](#). If one bit ECC error detected, the return value could be passed to the function [CSI2RX_GetEccBitErrorPosition](#) to get the position of the ECC error bit.

Example:

```
uint32_t bitError;
uint32_t bitErrorPosition;

bitError = CSI2RX_GetBitError(MIPI_CSI2RX);

if (kCSI2RX_BitErrorEccTwoBit & bitError)
{
    Two bits error;
}
else if (kCSI2RX_BitErrorEccOneBit & bitError)
{
    One bits error;
    bitErrorPosition = CSI2RX_GetEccBitErrorPosition(bitError);
}
```

Parameters

<i>base</i>	CSI2RX peripheral address.
-------------	----------------------------

Returns

The RX bit error status.

44.5.4 static uint32_t CSI2RX_GetEccBitErrorPosition (uint32_t *bitError*) [inline], [static]

If [CSI2RX_GetBitError](#) detects ECC one bit error, this function could extract the error bit position from the return value of [CSI2RX_GetBitError](#).

Parameters

<i>bitError</i>	The bit error returned by CSI2RX_GetBitError .
-----------------	--

Returns

The position of error bit.

44.5.5 static uint32_t CSI2RX_GetUlpsStatus (MIPI_CSI2RX_Type * *base*) [inline], [static]

Example to check whether data lane 0 is in ULPS status.

```
uint32_t status = CSI2RX_GetUlpsStatus(MIPI_CSI2RX);

if (kCSI2RX_DataLane0Ulps & status)
{
    Data lane 0 is in ULPS status.
}
```

Parameters

<i>base</i>	CSI2RX peripheral address.
-------------	----------------------------

Returns

The MIPI CSI2RX D-PHY ULPS status, it is OR'ed value or [_csi2rx_ulps_status](#).

44.5.6 static uint32_t CSI2RX_GetPpiErrorDataLanes (MIPI_CSI2RX_Type * *base*, csi2rx_ppi_error_t *errorType*) [inline], [static]

This function checks the PPI error occurred on which data lanes, the returned value is OR'ed value of [csi2rx_ppi_error_t](#). For example, if the ErrSotHS is detected, to check the ErrSotHS occurred on which data lanes, use like this:

```
uint32_t errorDataLanes = CSI2RX_GetPpiErrorDataLanes (MIPI_CSI2RX,
    kCSI2RX_PpiErrorSotHS);

if (kCSI2RX_DataLane0 & errorDataLanes)
{
    ErrSotHS occurred on data lane 0.
}

if (kCSI2RX_DataLane1 & errorDataLanes)
{
    ErrSotHS occurred on data lane 1.
}
```

Parameters

<i>base</i>	CSI2RX peripheral address.
<i>errorType</i>	What kind of error to check.

Returns

The data lane mask that error *errorType* occurred.

44.5.7 static void CSI2RX_EnableInterrupts (MIPI_CSI2RX_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the MIPI CSI2RX interrupts. The interrupts to enable are passed in as an OR'ed value of [_csi2rx_interrupt](#). For example, to enable one bit and two bit ECC error interrupts, use like this:

```
CSI2RX_EnableInterrupts (MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError |
    kCSI2RX_InterruptEccTwoBitError);
```

Parameters

<i>base</i>	CSI2RX peripheral address.
<i>mask</i>	OR'ed value of _csi2rx_interrupt .

44.5.8 static void CSI2RX_DisableInterrupts (MIPI_CSI2RX_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the MIPI CSI2RX interrupts. The interrupts to disable are passed in as an OR'ed value of [_csi2rx_interrupt](#). For example, to disable one bit and two bit ECC error interrupts, use like this:

```
CSI2RX_DisableInterrupts(MIPI_CSI2RX, kCSI2RX_InterruptEccOneBitError |
    kCSI2RX_InterruptEccTwoBitError);
```

Parameters

<i>base</i>	CSI2RX peripheral address.
<i>mask</i>	OR'ed value of _csi2rx_interrupt .

44.5.9 static uint32_t CSI2RX_GetInterruptStatus (MIPI_CSI2RX_Type * *base*) [inline], [static]

This function returns the MIPI CSI2RX interrupts status as an OR'ed value of [_csi2rx_interrupt](#).

Parameters

<i>base</i>	CSI2RX peripheral address.
-------------	----------------------------

Returns

OR'ed value of [_csi2rx_interrupt](#).

Chapter 45

MIPI_DSI: MIPI DSI Host Controller

45.1 Overview

The MCUXpresso SDK provides a peripheral driver for the MIPI DSI

Modules

- [MIPI DSI Driver](#)

The MIPI DSI driver supports both video mode and command mode.

45.2 MIPI DSI Driver

The MIPI DSI driver supports both video mode and command mode.

45.2.1 Overview

For both modes, first call [DSI_Init](#) and [DSI_InitDphy](#) to initialize the module and enable the D-PHY. The DSI driver provides function [DSI_GetDphyDefaultConfig](#) to help with the D-PHY timing parameter calculation. With the input txHsBitClk frequency and txEscClk frequency, the function can generate the timing parameters based on the D-PHY specification. The user can use the parameter directly, or change them according to the special device.

For the command mode, DSI driver provides polling method and interrupt method for the data transfer. At the same time, there are also small functional APIs so that user can construct them for their special purpose.

When the peripheral is configured through command mode, the video mode can be started by [DSI_SetDpiConfig](#).

45.2.2 Command mode data transfer

DSI driver provides polling method and interrupt method for the command mode data transfer, they are [DSI_TransferBlocking](#) and [DSI_TransferNonBlocking](#). The transfer is specified by the structure [dsi_transfer_t](#).

There are two ways to construct the [dsi_transfer_t](#).

1. Include the DSC command in TX data array. In this case, the DSC command is the first byte of TX data array. The parameter `sendDscCmd` is set to false, the `dscCmd` is not used.
2. The DSC command is not in TX data array, but specified by parameter `dscCmd`. In this case, the parameter `sendDscCmd` is set to true, the `dscCmd` is the DSC command to send. The TX data array is sent after `dscCmd`.

There is an example that send DSC command `set_column_address (0x2A)`. The two methods are actually the same.

Method 1: Include DSC command in TX data array.

```
dsi_transfer_t dsiXfer = {0};
uint8_t txData[4];

dsiXfer.virtualChannel = 0;
dsiXfer.txDataType     = kDSI_TxDataDcsLongWr;
dsiXfer.txDataSize     = 4;
dsiXfer.txData         = txData;
dsiXfer.sendDscCmd     = false;
dsiXfer.dscCmd         = 0; /* Not used. */

txData[0] = 0x2A;
txData[1] = (startX >> 8U) & 0xFFU;
txData[2] = startX & 0xFFU;
```

```
txData[3] = (endX >> 8U) & 0xFFU;
txData[4] = endX & 0xFFU;

DSI_TransferBlocking(MIPI_DSI, &dsiXfer);
```

Method 2: Don't include DSC command in TX data array.

```
dsi_transfer_t dsiXfer = {0};
uint8_t txData[5];

dsiXfer.virtualChannel = 0;
dsiXfer.txDataType      = kDSI_TxDataDcsLongWr;
dsiXfer.txDataSize      = 5;
dsiXfer.txData          = txData;
dsiXfer.sendDscCmd      = true;
dsiXfer.dscCmd          = 0x2A;

txData[0] = (startX >> 8U) & 0xFFU;
txData[1] = startX & 0xFFU;
txData[2] = (endX >> 8U) & 0xFFU;
txData[3] = endX & 0xFFU;

DSI_TransferBlocking(MIPI_DSI, &dsiXfer);
```

Data Structures

- struct [MIPI_DSI_Type](#)
MIPI DSI structure definition. [More...](#)
- struct [dsi_config_t](#)
MIPI DSI controller configuration. [More...](#)
- struct [dsi_dpi_config_t](#)
MIPI DSI controller DPI interface configuration. [More...](#)
- struct [dsi_dphy_config_t](#)
MIPI DSI D-PHY configuration. [More...](#)
- struct [dsi_transfer_t](#)
Structure for the data transfer. [More...](#)
- struct [dsi_handle_t](#)
MIPI DSI transfer handle structure. [More...](#)

Typedefs

- typedef void(* [dsi_callback_t](#))(const [MIPI_DSI_Type](#) *base, dsi_handle_t *handle, [status_t](#) status, void *userData)
MIPI DSI callback for finished transfer.

Enumerations

- enum {
 [kStatus_DSI_Busy](#) = MAKE_STATUS(kStatusGroup_MIPI_DSI, 0),
 [kStatus_DSI_RxDataError](#) = MAKE_STATUS(kStatusGroup_MIPI_DSI, 1),
 [kStatus_DSI_ErrorReportReceived](#) = MAKE_STATUS(kStatusGroup_MIPI_DSI, 2),

```
kStatus_DSI_NotSupported = MAKE_STATUS(kStatusGroup_MIPIDSI, 3) }
```

Error codes for the MIPI DSI driver.

- enum dsi_dpi_color_coding_t {
kDSI_Dpi16BitConfig1 = 0U,
kDSI_Dpi16BitConfig2 = 1U,
kDSI_Dpi16BitConfig3 = 2U,
kDSI_Dpi18BitConfig1 = 3U,
kDSI_Dpi18BitConfig2 = 4U,
kDSI_Dpi24Bit = 5U }

MIPI DPI interface color coding.

- enum dsi_dpi_pixel_packet_t {
kDSI_PixelPacket16Bit = 0U,
kDSI_PixelPacket18Bit = 1U,
kDSI_PixelPacket18BitLoosely = 2U,
kDSI_PixelPacket24Bit = 3U }

MIPI DSI pixel packet type send through DPI interface.

- enum {
kDSI_DpiVsyncActiveLow = 0U,
kDSI_DpiHsyncActiveLow = 0U,
kDSI_DpiVsyncActiveHigh = (1U << 0U),
kDSI_DpiHsyncActiveHigh = (1U << 1U) }

_dsi_dpi_polarity_flag DPI signal polarity.

- enum dsi_dpi_video_mode_t {
kDSI_DpiNonBurstWithSyncPulse = 0U,
kDSI_DpiNonBurstWithSyncEvent = 1U,
kDSI_DpiBurst = 2U }

DPI video mode.

- enum dsi_dpi_bllp_mode_t {
kDSI_DpiBllpLowPower,
kDSI_DpiBllpBlanking,
kDSI_DpiBllpNull }

Behavior in BLLP (Blanking or Low-Power Interval).

- enum {
kDSI_ApbNotIdle = (1U << 0U),
kDSI_ApbTxDone = (1U << 1U),
kDSI_ApbRxControl = (1U << 2U),
kDSI_ApbTxOverflow = (1U << 3U),
kDSI_ApbTxUnderflow = (1U << 4U),
kDSI_ApbRxOverflow = (1U << 5U),
kDSI_ApbRxUnderflow = (1U << 6U),
kDSI_ApbRxHeaderReceived = (1U << 7U),
kDSI_ApbRxPacketReceived = (1U << 8U) }

_dsi_apb_status Status of APB to packet interface.

- enum {

```

kDSI_RxErrorEccOneBit = (1U << 0U),
kDSI_RxErrorEccMultiBit = (1U << 1U),
kDSI_RxErrorCrc = (1U << 7U),
kDSI_RxErrorHtxTo = (1U << 8U),
kDSI_RxErrorLrxTo = (1U << 9U),
kDSI_RxErrorBtaTo = (1U << 10U) }

```

_dsi_rx_error_status Host receive error status.

- enum `_dsi_host_status` {


```

kDSI_HostSoTError = (1U << 0U),
kDSI_HostSoTSyncError = (1U << 1U),
kDSI_HostEoTSyncError = (1U << 2U),
kDSI_HostEscEntryCmdError = (1U << 3U),
kDSI_HostLpTxSyncError = (1U << 4U),
kDSI_HostPeriphToError = (1U << 5U),
kDSI_HostFalseControlError = (1U << 6U),
kDSI_HostContentionDetected = (1U << 7U),
kDSI_HostEccErrorOneBit = (1U << 8U),
kDSI_HostEccErrorMultiBit = (1U << 9U),
kDSI_HostChecksumError = (1U << 10U),
kDSI_HostInvalidDataType = (1U << 11U),
kDSI_HostInvalidVcId = (1U << 12U),
kDSI_HostInvalidTxLength = (1U << 13U),
kDSI_HostProtocolViolation = (1U << 15U),
kDSI_HostResetTriggerReceived = (1U << 16U),
kDSI_HostTearTriggerReceived = (1U << 17U),
kDSI_HostAckTriggerReceived = (1U << 18U) }

```

DSI host controller status (status_out)

- enum {

```

kDSI_InterruptGroup1ApbNotIdle = (1U << 0U),
kDSI_InterruptGroup1ApbTxDone = (1U << 1U),
kDSI_InterruptGroup1ApbRxControl = (1U << 2U),
kDSI_InterruptGroup1ApbTxOverflow = (1U << 3U),
kDSI_InterruptGroup1ApbTxUnderflow = (1U << 4U),
kDSI_InterruptGroup1ApbRxOverflow = (1U << 5U),
kDSI_InterruptGroup1ApbRxUnderflow = (1U << 6U),
kDSI_InterruptGroup1ApbRxHeaderReceived = (1U << 7U),
kDSI_InterruptGroup1ApbRxPacketReceived = (1U << 8U),
kDSI_InterruptGroup1SoTError = (1U << 9U),
kDSI_InterruptGroup1SoTSyncError = (1U << 10U),
kDSI_InterruptGroup1EoTSyncError = (1U << 11U),
kDSI_InterruptGroup1EscEntryCmdError = (1U << 12U),
kDSI_InterruptGroup1LpTxSyncError = (1U << 13U),
kDSI_InterruptGroup1PeriphToError = (1U << 14U),
kDSI_InterruptGroup1FalseControlError = (1U << 15U),
kDSI_InterruptGroup1ContentionDetected = (1U << 16U),
kDSI_InterruptGroup1EccErrorOneBit = (1U << 17U),
kDSI_InterruptGroup1EccErrorMultiBit = (1U << 18U),
kDSI_InterruptGroup1ChecksumError = (1U << 19U),
kDSI_InterruptGroup1InvalidDataType = (1U << 20U),
kDSI_InterruptGroup1InvalidVcId = (1U << 21U),
kDSI_InterruptGroup1InvalidTxLength = (1U << 22U),
kDSI_InterruptGroup1ProtocalViolation = (1U << 24U),
kDSI_InterruptGroup1ResetTriggerReceived = (1U << 25U),
kDSI_InterruptGroup1TearTriggerReceived = (1U << 26U),
kDSI_InterruptGroup1AckTriggerReceived = (1U << 27U),
kDSI_InterruptGroup1HtxTo = (1U << 29U),
kDSI_InterruptGroup1LrxTo = (1U << 30U),
kDSI_InterruptGroup1BtaTo = (1U << 31U),
kDSI_InterruptGroup2EccOneBit = (1U << 0U),
kDSI_InterruptGroup2EccMultiBit = (1U << 1U),
kDSI_InterruptGroup2CrcError = (1U << 2U) }

```

_dsi_interrupt DSI interrupt.

- enum `dsi_tx_data_type_t` {


```

kDSI_TxDataVsyncStart = 0x01U,
kDSI_TxDataVsyncEnd = 0x11U,
kDSI_TxDataHsyncStart = 0x21U,
kDSI_TxDataHsyncEnd = 0x31U,
kDSI_TxDataEoTp = 0x08U,
kDSI_TxDataCmOff = 0x02U,
kDSI_TxDataCmOn = 0x12U,
kDSI_TxDataShutDownPeriph = 0x22U,
kDSI_TxDataTurnOnPeriph = 0x32U,
kDSI_TxDataGenShortWrNoParam = 0x03U,
kDSI_TxDataGenShortWrOneParam = 0x13U,
kDSI_TxDataGenShortWrTwoParam = 0x23U,
kDSI_TxDataGenShortRdNoParam = 0x04U,
kDSI_TxDataGenShortRdOneParam = 0x14U,
kDSI_TxDataGenShortRdTwoParam = 0x24U,
kDSI_TxDataDcsShortWrNoParam = 0x05U,
kDSI_TxDataDcsShortWrOneParam = 0x15U,
kDSI_TxDataDcsShortRdNoParam = 0x06U,
kDSI_TxDataSetMaxReturnPktSize = 0x37U,
kDSI_TxDataNull = 0x09U,
kDSI_TxDataBlanking = 0x19U,
kDSI_TxDataGenLongWr = 0x29U,
kDSI_TxDataDcsLongWr = 0x39U,
kDSI_TxDataLooselyPackedPixel20BitYCbCr = 0x0CU,
kDSI_TxDataPackedPixel24BitYCbCr = 0x1CU,
kDSI_TxDataPackedPixel16BitYCbCr = 0x2CU,
kDSI_TxDataPackedPixel30BitRGB = 0x0DU,
kDSI_TxDataPackedPixel36BitRGB = 0x1DU,
kDSI_TxDataPackedPixel12BitYCrCb = 0x3DU,
kDSI_TxDataPackedPixel16BitRGB = 0x0EU,
kDSI_TxDataPackedPixel18BitRGB = 0x1EU,
kDSI_TxDataLooselyPackedPixel18BitRGB = 0x2EU,
kDSI_TxDataPackedPixel24BitRGB = 0x3EU }

```

DSI TX data type.

- enum dsi_rx_data_type_t {


```

kDSI_RxDataAckAndErrorReport = 0x02U,
kDSI_RxDataEoTp = 0x08U,
kDSI_RxDataGenShortRdResponseOneByte = 0x11U,
kDSI_RxDataGenShortRdResponseTwoByte = 0x12U,
kDSI_RxDataGenLongRdResponse = 0x1AU,
kDSI_RxDataDcsLongRdResponse = 0x1CU,
kDSI_RxDataDcsShortRdResponseOneByte = 0x21U,
kDSI_RxDataDcsShortRdResponseTwoByte = 0x22U }

```

DSI RX data type.

- enum {

```

kDSI_TransferUseHighSpeed = (1U << 0U),
kDSI_TransferPerformBTA = (1U << 1U) }
_dsi_transfer_flags DSI transfer control flags.

```

Driver version

- #define **FSL_MIPI_DSI_DRIVER_VERSION** ([MAKE_VERSION](#)(2, 2, 1))

MIPI_DSI host initialization.

- void [DSI_Init](#) (const [MIPI_DSI_Type](#) *base, const [dsi_config_t](#) *config)
Initializes an MIPI DSI host with the user configuration.
- void [DSI_Deinit](#) (const [MIPI_DSI_Type](#) *base)
Deinitializes an MIPI DSI host.
- void [DSI_GetDefaultConfig](#) ([dsi_config_t](#) *config)
Get the default configuration to initialize the MIPI DSI host.

DPI interface

- void [DSI_SetDpiConfig](#) (const [MIPI_DSI_Type](#) *base, const [dsi_dpi_config_t](#) *config, uint8_t numLanes, uint32_t dpiPixelClkFreq_Hz, uint32_t dsiHsBitClkFreq_Hz)
Configure the DPI interface core.

D-PHY configuration.

- uint32_t [DSI_InitDphy](#) (const [MIPI_DSI_Type](#) *base, const [dsi_dphy_config_t](#) *config, uint32_t refClkFreq_Hz)
Initializes the D-PHY.
- void [DSI_DeinitDphy](#) (const [MIPI_DSI_Type](#) *base)
Deinitializes the D-PHY.
- void [DSI_GetDphyDefaultConfig](#) ([dsi_dphy_config_t](#) *config, uint32_t txHsBitClk_Hz, uint32_t txEscClk_Hz)
Get the default D-PHY configuration.

Interrupts

- static void [DSI_EnableInterrupts](#) (const [MIPI_DSI_Type](#) *base, uint32_t intGroup1, uint32_t intGroup2)
Enable the interrupts.
- static void [DSI_DisableInterrupts](#) (const [MIPI_DSI_Type](#) *base, uint32_t intGroup1, uint32_t intGroup2)
Disable the interrupts.

- static void **DSI_GetAndClearInterruptStatus** (const **MIPI_DSI_Type** *base, uint32_t *intGroup1, uint32_t *intGroup2)
Get and clear the interrupt status.

MIPI DSI APB

- void **DSI_SetApbPacketControl** (const **MIPI_DSI_Type** *base, uint16_t wordCount, uint8_t virtualChannel, **dsi_tx_data_type_t** dataType, uint8_t flags)
Configure the APB packet to send.
- void **DSI_WriteApbTxPayload** (const **MIPI_DSI_Type** *base, const uint8_t *payload, uint16_t payloadSize)
Fill the long APB packet payload.
- void **DSI_WriteApbTxPayloadExt** (const **MIPI_DSI_Type** *base, const uint8_t *payload, uint16_t payloadSize, bool sendDscCmd, uint8_t dscCmd)
Extended function to fill the payload to TX FIFO.
- void **DSI_ReadApbRxPayload** (const **MIPI_DSI_Type** *base, uint8_t *payload, uint16_t payloadSize)
Read the long APB packet payload.
- static void **DSI_SendApbPacket** (const **MIPI_DSI_Type** *base)
Trigger the controller to send out APB packet.
- static uint32_t **DSI_GetApbStatus** (const **MIPI_DSI_Type** *base)
Get the APB status.
- static uint32_t **DSI_GetRxErrorStatus** (const **MIPI_DSI_Type** *base)
Get the error status during data transfer.
- static uint8_t **DSI_GetEccRxErrorPosition** (uint32_t rxErrorStatus)
Get the one-bit RX ECC error position.
- static uint32_t **DSI_GetAndClearHostStatus** (const **MIPI_DSI_Type** *base)
Get and clear the DSI host status.
- static uint32_t **DSI_GetRxPacketHeader** (const **MIPI_DSI_Type** *base)
Get the RX packet header.
- static **dsi_rx_data_type_t** **DSI_GetRxPacketType** (uint32_t rxPktHeader)
Extract the RX packet type from the packet header.
- static uint16_t **DSI_GetRxPacketWordCount** (uint32_t rxPktHeader)
Extract the RX packet word count from the packet header.
- static uint8_t **DSI_GetRxPacketVirtualChannel** (uint32_t rxPktHeader)
Extract the RX packet virtual channel from the packet header.
- **status_t** **DSI_TransferBlocking** (const **MIPI_DSI_Type** *base, **dsi_transfer_t** *xfer)
APB data transfer using blocking method.

Transactional

- **status_t** **DSI_TransferCreateHandle** (const **MIPI_DSI_Type** *base, **dsi_handle_t** *handle, **dsi_callback_t** callback, void *userData)
Create the MIPI DSI handle.
- **status_t** **DSI_TransferNonBlocking** (const **MIPI_DSI_Type** *base, **dsi_handle_t** *handle, **dsi_transfer_t** *xfer)
APB data transfer using interrupt method.

- void [DSI_TransferAbort](#) (const [MIPI_DSI_Type](#) *base, dsi_handle_t *handle)
Abort current APB data transfer.
- void [DSI_TransferHandleIRQ](#) (const [MIPI_DSI_Type](#) *base, dsi_handle_t *handle)
Interrupt handler for the DSI.

45.2.3 Data Structure Documentation

45.2.3.1 struct MIPI_DSI_Type

Data Fields

- DSI_HOST_Type * [host](#)
Pointer to HOST registers.
- DSI_HOST_APB_PKT_IF_Type * [apb](#)
Pointer to APB registers.
- DSI_HOST_DPI_INTFC_Type * [dpi](#)
Pointer to DPI registers.
- DSI_HOST_NXP_FDSOI28_DPHY_INTFC_Type * [dphy](#)
Pointer to DPHY registers.

Field Documentation

- (1) [DSI_HOST_Type](#)* [MIPI_DSI_Type::host](#)
- (2) [DSI_HOST_APB_PKT_IF_Type](#)* [MIPI_DSI_Type::apb](#)
- (3) [DSI_HOST_DPI_INTFC_Type](#)* [MIPI_DSI_Type::dpi](#)
- (4) [DSI_HOST_NXP_FDSOI28_DPHY_INTFC_Type](#)* [MIPI_DSI_Type::dphy](#)

45.2.3.2 struct dsi_config_t

Data Fields

- uint8_t [numLanes](#)
Number of lanes.
- bool [enableNonContinuousHsClk](#)
In enabled, the high speed clock will enter low power mode between transmissions.
- bool [enableTxUlps](#)
Enable the TX ULPS.
- bool [autoInsertEoTp](#)
Insert an EoTp short package when switching from HS to LP.
- uint8_t [numExtraEoTp](#)
How many extra EoTp to send after the end of a packet.
- uint32_t [htxTo_ByteClk](#)
HS TX timeout count (HTX_TO) in byte clock.
- uint32_t [lrxHostTo_ByteClk](#)
LP RX host timeout count (LRX-H_TO) in byte clock.
- uint32_t [btaTo_ByteClk](#)

Bus turn around timeout count (TA_TO) in byte clock.

Field Documentation

- (1) `uint8_t dsi_config_t::numLanes`
- (2) `bool dsi_config_t::enableNonContinuousHsClk`
- (3) `bool dsi_config_t::enableTxUlps`
- (4) `bool dsi_config_t::autoInsertEoTp`
- (5) `uint8_t dsi_config_t::numExtraEoTp`
- (6) `uint32_t dsi_config_t::htxTo_ByteClk`
- (7) `uint32_t dsi_config_t::lrxHostTo_ByteClk`
- (8) `uint32_t dsi_config_t::btaTo_ByteClk`

45.2.3.3 struct dsi_dpi_config_t

Data Fields

- `uint16_t pixelPayloadSize`
Maximum number of pixels that should be sent as one DSI packet.
- `dsi_dpi_color_coding_t dpiColorCoding`
DPI color coding.
- `dsi_dpi_pixel_packet_t pixelPacket`
Pixel packet format.
- `dsi_dpi_video_mode_t videoMode`
Video mode.
- `dsi_dpi_bllp_mode_t bllpMode`
Behavior in BLLP.
- `uint8_t polarityFlags`
OR'ed value of _dsi_dpi_polarity_flag controls signal polarity.
- `uint16_t hfp`
Horizontal front porch, in dpi pixel clock.
- `uint16_t hbp`
Horizontal back porch, in dpi pixel clock.
- `uint16_t hsw`
Horizontal sync width, in dpi pixel clock.
- `uint8_t vfp`
Number of lines in vertical front porch.
- `uint8_t vbp`
Number of lines in vertical back porch.
- `uint16_t panelHeight`
Line number in vertical active area.
- `uint8_t virtualChannel`
Virtual channel.

Field Documentation

(1) `uint16_t dsi_dpi_config_t::pixelPayloadSize`

Recommended that the line size (in pixels) is evenly divisible by this parameter.

(2) `dsi_dpi_color_coding_t dsi_dpi_config_t::dpiColorCoding`(3) `dsi_dpi_pixel_packet_t dsi_dpi_config_t::pixelPacket`(4) `dsi_dpi_video_mode_t dsi_dpi_config_t::videoMode`(5) `dsi_dpi_bllp_mode_t dsi_dpi_config_t::bllpMode`(6) `uint8_t dsi_dpi_config_t::polarityFlags`(7) `uint16_t dsi_dpi_config_t::hfp`(8) `uint16_t dsi_dpi_config_t::hbp`(9) `uint16_t dsi_dpi_config_t::hsw`(10) `uint8_t dsi_dpi_config_t::vfp`(11) `uint8_t dsi_dpi_config_t::vbp`(12) `uint16_t dsi_dpi_config_t::panelHeight`(13) `uint8_t dsi_dpi_config_t::virtualChannel`45.2.3.4 `struct dsi_dphy_config_t`

Data Fields

- `uint32_t txHsBitClk_Hz`
The generated HS TX bit clock in Hz.
- `uint8_t tClkPre_ByteClk`
TLPX + TCLK-PREPARE + TCLK-ZERO + TCLK-PRE in byte clock.
- `uint8_t tClkPost_ByteClk`
TCLK-POST + T_CLK-TRAIL in byte clock.
- `uint8_t tHsExit_ByteClk`
THS-EXIT in byte clock.
- `uint32_t tWakeup_EscClk`
Number of clk_esc clock periods to keep a clock or data lane in Mark-1 state after exiting ULPS.
- `uint8_t tHsPrepare_HalfEscClk`
THS-PREPARE in clk_esc/2.
- `uint8_t tClkPrepare_HalfEscClk`
TCLK-PREPARE in clk_esc/2.
- `uint8_t tHsZero_ByteClk`
THS-ZERO in clk_byte.
- `uint8_t tClkZero_ByteClk`

- $TCLK-ZERO$ in clk_byte .
uint8_t tHsTrail_ByteClk
 $THS-TRAIL + 4*UI$ in clk_byte .
- uint8_t tClkTrail_ByteClk
 $TCLK-TRAIL + 4*UI$ in clk_byte .

Field Documentation

(1) uint32_t dsi_dphy_config_t::txHsBitClk_Hz

(2) uint8_t dsi_dphy_config_t::tClkPre_ByteClk

Set how long the controller will wait after enabling clock lane for HS before enabling data lanes for HS.

(3) uint8_t dsi_dphy_config_t::tClkPost_ByteClk

Set how long the controller will wait before putting clock lane into LP mode after data lanes detected in stop state.

(4) uint8_t dsi_dphy_config_t::tHsExit_ByteClk

Set how long the controller will wait after the clock lane has been put into LP mode before enabling clock lane for HS again.

(5) uint32_t dsi_dphy_config_t::tWakeup_EscClk

(6) uint8_t dsi_dphy_config_t::tHsPrepare_HalfEscClk

Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3, 4, 5.

(7) uint8_t dsi_dphy_config_t::tClkPrepare_HalfEscClk

Set how long to drive the LP-00 state before HS transmissions, available values are 2, 3.

(8) uint8_t dsi_dphy_config_t::tHsZero_ByteClk

Set how long that controller drives data lane HS-0 state before transmit the Sync sequence. Available values are 6, 7, ..., 37.

(9) uint8_t dsi_dphy_config_t::tClkZero_ByteClk

Set how long that controller drives clock lane HS-0 state before transmit the Sync sequence. Available values are 3, 4, ..., 66.

(10) uint8_t dsi_dphy_config_t::tHsTrail_ByteClk

Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

(11) uint8_t dsi_dphy_config_t::tClkTrail_ByteClk

Set the time of the flipped differential state after last payload data bit of HS transmission burst. Available values are 0, 1, ..., 15.

45.2.3.5 struct dsi_transfer_t

Data Fields

- uint8_t [virtualChannel](#)
Virtual channel.
- [dsi_tx_data_type_t](#) txDataType
TX data type.
- uint8_t [flags](#)
Flags to control the transfer, see [_dsi_transfer_flags](#).
- const uint8_t * [txData](#)
The TX data buffer.
- uint8_t * [rxData](#)
The TX data buffer.
- uint16_t [txDataSize](#)
Size of the TX data.
- uint16_t [rxDataSize](#)
Size of the RX data.
- bool [sendDscCmd](#)
If set to true, the DSC command is specified by [dscCmd](#), otherwise the DSC command is included in the [txData](#).
- uint8_t [dscCmd](#)
The DSC command to send, only valid when [sendDscCmd](#) is true.

Field Documentation

- (1) uint8_t dsi_transfer_t::virtualChannel
- (2) dsi_tx_data_type_t dsi_transfer_t::txDataType
- (3) uint8_t dsi_transfer_t::flags
- (4) const uint8_t* dsi_transfer_t::txData
- (5) uint8_t* dsi_transfer_t::rxData
- (6) uint16_t dsi_transfer_t::txDataSize
- (7) uint16_t dsi_transfer_t::rxDataSize
- (8) bool dsi_transfer_t::sendDscCmd
- (9) uint8_t dsi_transfer_t::dscCmd

45.2.3.6 struct _dsi_handle

MIPI DSI transfer handle.

Data Fields

- volatile bool [isBusy](#)
MIPI DSI is busy with APB data transfer.
- [dsi_transfer_t](#) xfer
Transfer information.
- [dsi_callback_t](#) callback
DSI callback.
- void * [userData](#)
Callback parameter.
- const [MIPI_DSI_Type](#) * [dsi](#)
Pointer to MIPI DSI peripheral.

Field Documentation

- (1) volatile bool [dsi_handle_t::isBusy](#)
- (2) [dsi_transfer_t](#) [dsi_handle_t::xfer](#)
- (3) const [MIPI_DSI_Type](#)* [dsi_handle_t::dsi](#)

45.2.4 Typedef Documentation

45.2.4.1 typedef void(* [dsi_callback_t](#))(const [MIPI_DSI_Type](#) *base, [dsi_handle_t](#) *handle, [status_t](#) status, void *userData)

When transfer finished, one of these status values will be passed to the user:

- [kStatus_Success](#) Data transfer finished with no error.
- [kStatus_Timeout](#) Transfer failed because of timeout.
- [kStatus_DSI_RxDataError](#) RX data error, user could use [DSI_GetRxErrorStatus](#) to check the error details.
- [kStatus_DSI_ErrorReportReceived](#) Error Report packet received, user could use [DSI_GetAndClear-HostStatus](#) to check the error report status.
- [kStatus_Fail](#) Transfer failed for other reasons.

45.2.5 Enumeration Type Documentation

45.2.5.1 anonymous enum

Enumerator

kStatus_DSI_Busy DSI is busy.

kStatus_DSI_RxDataError Read data error.

kStatus_DSI_ErrorReportReceived Error report package received.

kStatus_DSI_NotSupported The transfer type not supported.

45.2.5.2 enum dsi_dpi_color_coding_t

Enumerator

kDSI_Dpi16BitConfig1 16-bit configuration 1. RGB565: XXXXXXXX_RRRRRGGG_GGGBB-BBB.

kDSI_Dpi16BitConfig2 16-bit configuration 2. RGB565: XXXRRRRR_XXGGGGGG_XXXBB-BBB.

kDSI_Dpi16BitConfig3 16-bit configuration 3. RGB565: XXRRRRRX_XXGGGGGG_XXBBB-BBX.

kDSI_Dpi18BitConfig1 18-bit configuration 1. RGB666: XXXXXXRR_RRRRGGGG_GGBBB-BBB.

kDSI_Dpi18BitConfig2 18-bit configuration 2. RGB666: XXRRRRRR_XXGGGGGG_XXBBB-BBB.

kDSI_Dpi24Bit 24-bit.

45.2.5.3 enum dsi_dpi_pixel_packet_t

Enumerator

kDSI_PixelPacket16Bit 16 bit RGB565.

kDSI_PixelPacket18Bit 18 bit RGB666 packed.

kDSI_PixelPacket18BitLoosely 18 bit RGB666 loosely packed into three bytes.

kDSI_PixelPacket24Bit 24 bit RGB888, each pixel uses three bytes.

45.2.5.4 anonymous enum

Enumerator

kDSI_DpiVsyncActiveLow VSYNC active low.

kDSI_DpiHsyncActiveLow HSYNC active low.

kDSI_DpiVsyncActiveHigh VSYNC active high.

kDSI_DpiHsyncActiveHigh HSYNC active high.

45.2.5.5 enum dsi_dpi_video_mode_t

Enumerator

kDSI_DpiNonBurstWithSyncPulse Non-Burst mode with Sync Pulses.

kDSI_DpiNonBurstWithSyncEvent Non-Burst mode with Sync Events.
kDSI_DpiBurst Burst mode.

45.2.5.6 enum dsi_dpi_bllp_mode_t

Enumerator

kDSI_DpiBllpLowPower LP mode used in BLLP periods.
kDSI_DpiBllpBlanking Blanking packets used in BLLP periods.
kDSI_DpiBllpNull Null packets used in BLLP periods.

45.2.5.7 anonymous enum

Enumerator

kDSI_ApbNotIdle State machine not idle.
kDSI_ApbTxDone Tx packet done.
kDSI_ApbRxControl DPHY direction 0 - tx had control, 1 - rx has control.
kDSI_ApbTxOverflow TX fifo overflow.
kDSI_ApbTxUnderflow TX fifo underflow.
kDSI_ApbRxOverflow RX fifo overflow.
kDSI_ApbRxUnderflow RX fifo underflow.
kDSI_ApbRxHeaderReceived RX packet header has been received.
kDSI_ApbRxPacketReceived All RX packet payload data has been received.

45.2.5.8 anonymous enum

Enumerator

kDSI_RxErrorEccOneBit ECC single bit error detected.
kDSI_RxErrorEccMultiBit ECC multi bit error detected.
kDSI_RxErrorCrc CRC error detected.
kDSI_RxErrorHtxTo High Speed forward TX timeout detected.
kDSI_RxErrorLrxTo Reverse Low power data receive timeout detected.
kDSI_RxErrorBtaTo BTA timeout detected.

45.2.5.9 enum _dsi_host_status

Enumerator

kDSI_HostSoTError SoT error from peripheral error report.
kDSI_HostSoTSyncError SoT Sync error from peripheral error report.
kDSI_HostEoTSyncError EoT Sync error from peripheral error report.
kDSI_HostEscEntryCmdError Escape Mode Entry Command Error from peripheral error report.

kDSI_HostLpTxSyncError Low-power transmit Sync Error from peripheral error report.
kDSI_HostPeriphToError Peripheral timeout error from peripheral error report.
kDSI_HostFalseControlError False control error from peripheral error report.
kDSI_HostContentionDetected Contention detected from peripheral error report.
kDSI_HostEccErrorOneBit Single bit ECC error (corrected) from peripheral error report.
kDSI_HostEccErrorMultiBit Multi bit ECC error (not corrected) from peripheral error report.
kDSI_HostChecksumError Checksum error from peripheral error report.
kDSI_HostInvalidDataType DSI data type not recognized.
kDSI_HostInvalidVcId DSI VC ID invalid.
kDSI_HostInvalidTxLength Invalid transmission length.
kDSI_HostProtocolViolation DSI protocol violation.
kDSI_HostResetTriggerReceived Reset trigger received.
kDSI_HostTearTriggerReceived Tear effect trigger receive.
kDSI_HostAckTriggerReceived Acknowledge trigger message received.

45.2.5.10 anonymous enum

Enumerator

kDSI_InterruptGroup1ApbNotIdle State machine not idle.
kDSI_InterruptGroup1ApbTxDone Tx packet done.
kDSI_InterruptGroup1ApbRxControl DPHY direction 0 - tx control, 1 - rx control.
kDSI_InterruptGroup1ApbTxOverflow TX fifo overflow.
kDSI_InterruptGroup1ApbTxUnderflow TX fifo underflow.
kDSI_InterruptGroup1ApbRxOverflow RX fifo overflow.
kDSI_InterruptGroup1ApbRxUnderflow RX fifo underflow.
kDSI_InterruptGroup1ApbRxHeaderReceived RX packet header has been received.
kDSI_InterruptGroup1ApbRxPacketReceived All RX packet payload data has been received.
kDSI_InterruptGroup1SoTError SoT error from peripheral error report.
kDSI_InterruptGroup1SoTSyncError SoT Sync error from peripheral error report.
kDSI_InterruptGroup1EoTSyncError EoT Sync error from peripheral error report.
kDSI_InterruptGroup1EscEntryCmdError Escape Mode Entry Command Error from peripheral error report.
kDSI_InterruptGroup1LpTxSyncError Low-power transmit Sync Error from peripheral error report.
kDSI_InterruptGroup1PeriphToError Peripheral timeout error from peripheral error report.
kDSI_InterruptGroup1FalseControlError False control error from peripheral error report.
kDSI_InterruptGroup1ContentionDetected Contention detected from peripheral error report.
kDSI_InterruptGroup1EccErrorOneBit Single bit ECC error (corrected) from peripheral error report.
kDSI_InterruptGroup1EccErrorMultiBit Multi bit ECC error (not corrected) from peripheral error report.
kDSI_InterruptGroup1ChecksumError Checksum error from peripheral error report.
kDSI_InterruptGroup1InvalidDataType DSI data type not recognized.
kDSI_InterruptGroup1InvalidVcId DSI VC ID invalid.

kDSI_InterruptGroup1InvalidTxLength Invalid transmission length.
kDSI_InterruptGroup1ProtocolViolation DSI protocol violation.
kDSI_InterruptGroup1ResetTriggerReceived Reset trigger received.
kDSI_InterruptGroup1TearTriggerReceived Tear effect trigger receive.
kDSI_InterruptGroup1AckTriggerReceived Acknowledge trigger message received.
kDSI_InterruptGroup1HtxTo High speed TX timeout.
kDSI_InterruptGroup1LrxTo Low power RX timeout.
kDSI_InterruptGroup1BtaTo Host BTA timeout.
kDSI_InterruptGroup2EccOneBit Single bit ECC error.
kDSI_InterruptGroup2EccMultiBit Multi bit ECC error.
kDSI_InterruptGroup2CrcError CRC error.

45.2.5.11 enum dsi_tx_data_type_t

Enumerator

kDSI_TxDataVsyncStart V Sync start.
kDSI_TxDataVsyncEnd V Sync end.
kDSI_TxDataHsyncStart H Sync start.
kDSI_TxDataHsyncEnd H Sync end.
kDSI_TxDataEoTp End of transmission packet.
kDSI_TxDataCmOff Color mode off.
kDSI_TxDataCmOn Color mode on.
kDSI_TxDataShutDownPeriph Shut down peripheral.
kDSI_TxDataTurnOnPeriph Turn on peripheral.
kDSI_TxDataGenShortWrNoParam Generic Short WRITE, no parameters.
kDSI_TxDataGenShortWrOneParam Generic Short WRITE, one parameter.
kDSI_TxDataGenShortWrTwoParam Generic Short WRITE, two parameter.
kDSI_TxDataGenShortRdNoParam Generic Short READ, no parameters.
kDSI_TxDataGenShortRdOneParam Generic Short READ, one parameter.
kDSI_TxDataGenShortRdTwoParam Generic Short READ, two parameter.
kDSI_TxDataDcsShortWrNoParam DCS Short WRITE, no parameters.
kDSI_TxDataDcsShortWrOneParam DCS Short WRITE, one parameter.
kDSI_TxDataDcsShortRdNoParam DCS Short READ, no parameters.
kDSI_TxDataSetMaxReturnPktSize Set the Maximum Return Packet Size.
kDSI_TxDataNull Null Packet, no data.
kDSI_TxDataBlanking Blanking Packet, no data.
kDSI_TxDataGenLongWr Generic long write.
kDSI_TxDataDcsLongWr DCS Long Write/write_LUT Command Packet.
kDSI_TxDataLooselyPackedPixel20BitYCbCr Loosely Packed Pixel Stream, 20-bit YCbCr, 4:2:2 Format.
kDSI_TxDataPackedPixel24BitYCbCr Packed Pixel Stream, 24-bit YCbCr, 4:2:2 Format.
kDSI_TxDataPackedPixel16BitYCbCr Packed Pixel Stream, 16-bit YCbCr, 4:2:2 Format.
kDSI_TxDataPackedPixel30BitRGB Packed Pixel Stream, 30-bit RGB, 10-10-10 Format.

kDSI_TxDataPackedPixel36BitRGB Packed Pixel Stream, 36-bit RGB, 12-12-12 Format.
kDSI_TxDataPackedPixel12BitYCrCb Packed Pixel Stream, 12-bit YCbCr, 4:2:0 Format.
kDSI_TxDataPackedPixel16BitRGB Packed Pixel Stream, 16-bit RGB, 5-6-5 Format.
kDSI_TxDataPackedPixel18BitRGB Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.
kDSI_TxDataLooselyPackedPixel18BitRGB Loosely Packed Pixel Stream, 18-bit RGB, 6-6-6 Format.
kDSI_TxDataPackedPixel24BitRGB Packed Pixel Stream, 24-bit RGB, 8-8-8 Format.

45.2.5.12 enum dsi_rx_data_type_t

Enumerator

kDSI_RxDataAckAndErrorReport Acknowledge and Error Report.
kDSI_RxDataEoTp End of Transmission packet.
kDSI_RxDataGenShortRdResponseOneByte Generic Short READ Response, 1 byte returned.
kDSI_RxDataGenShortRdResponseTwoByte Generic Short READ Response, 2 byte returned.
kDSI_RxDataGenLongRdResponse Generic Long READ Response.
kDSI_RxDataDcsLongRdResponse DCS Long READ Response.
kDSI_RxDataDcsShortRdResponseOneByte DCS Short READ Response, 1 byte returned.
kDSI_RxDataDcsShortRdResponseTwoByte DCS Short READ Response, 2 byte returned.

45.2.5.13 anonymous enum

Enumerator

kDSI_TransferUseHighSpeed Use high speed mode or not.
kDSI_TransferPerformBTA Perform BTA or not.

45.2.6 Function Documentation

45.2.6.1 void DSI_Init (const MIPI_DSI_Type * *base*, const dsi_config_t * *config*)

This function initializes the MIPI DSI host with the configuration, it should be called first before other MIPI DSI driver functions.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

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

45.2.6.2 void DSI_Deinit (const MIPI_DSI_Type * *base*)

This function should be called after all bother MIPI DSI driver functions.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

45.2.6.3 void DSI_GetDefaultConfig (dsi_config_t * *config*)

The default value is:

```
config->numLanes = 4;
config->enableNonContinuousHsClk = false;
config->enableTxUlps = false;
config->autoInsertEoTp = true;
config->numExtraEoTp = 0;
config->htxTo_ByteClk = 0;
config->lrxHostTo_ByteClk = 0;
config->btaTo_ByteClk = 0;
```

Parameters

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

45.2.6.4 void DSI_SetDpiConfig (const MIPI_DSI_Type * *base*, const dsi_dpi_config_t * *config*, uint8_t *numLanes*, uint32_t *dpiPixelClkFreq_Hz*, uint32_t *dsiHsBitClkFreq_Hz*)

This function sets the DPI interface configuration, it should be used in video mode.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>config</i>	Pointer to the DPI interface configuration.

<i>numLanes</i>	Lane number, should be same with the setting in dsi_dpi_config_t .
<i>dpiPixelClk-Freq_Hz</i>	The DPI pixel clock frequency in Hz.
<i>dsiHsBitClk-Freq_Hz</i>	The DSI high speed bit clock frequency in Hz. It is the same with DPHY PLL output.

45.2.6.5 **uint32_t DSI_InitDphy (const MIPI_DSI_Type * *base*, const dsi_dphy_config_t * *config*, uint32_t *refClkFreq_Hz*)**

This function configures the D-PHY timing and setups the D-PHY PLL based on user configuration. The configuration structure could be got by the function [DSI_GetDphyDefaultConfig](#).

For some platforms there is not dedicated D-PHY PLL, indicated by the macro FSL_FEATURE_MIPI_DSI_NO_DPHY_PLL. For these platforms, the *refClkFreq_Hz* is useless.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>config</i>	Pointer to the D-PHY configuration.
<i>refClkFreq_Hz</i>	The REFCLK frequency in Hz.

Returns

The actual D-PHY PLL output frequency. If could not configure the PLL to the target frequency, the return value is 0.

45.2.6.6 **void DSI_DeinitDphy (const MIPI_DSI_Type * *base*)**

Power down the D-PHY PLL and shut down D-PHY.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

45.2.6.7 **void DSI_GetDphyDefaultConfig (dsi_dphy_config_t * *config*, uint32_t *txHsBitClk_Hz*, uint32_t *txEscClk_Hz*)**

Gets the default D-PHY configuration, the timing parameters are set according to D-PHY specification. User could use the configuration directly, or change some parameters according to the special device.

Parameters

<i>config</i>	Pointer to the D-PHY configuration.
<i>txHsBitClk_Hz</i>	High speed bit clock in Hz.
<i>txEscClk_Hz</i>	Esc clock in Hz.

45.2.6.8 static void DSI_EnableInterrupts (const MIPI_DSI_Type * *base*, uint32_t *intGroup1*, uint32_t *intGroup2*) [inline], [static]

The interrupts to enable are passed in as OR'ed mask value of _dsi_interrupt.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>intGroup1</i>	Interrupts to enable in group 1.
<i>intGroup2</i>	Interrupts to enable in group 2.

45.2.6.9 static void DSI_DisableInterrupts (const MIPI_DSI_Type * *base*, uint32_t *intGroup1*, uint32_t *intGroup2*) [inline], [static]

The interrupts to disable are passed in as OR'ed mask value of _dsi_interrupt.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>intGroup1</i>	Interrupts to disable in group 1.
<i>intGroup2</i>	Interrupts to disable in group 2.

45.2.6.10 static void DSI_GetAndClearInterruptStatus (const MIPI_DSI_Type * *base*, uint32_t * *intGroup1*, uint32_t * *intGroup2*) [inline], [static]

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

<i>intGroup1</i>	Group 1 interrupt status.
<i>intGroup2</i>	Group 2 interrupt status.

45.2.6.11 void DSI_SetApbPacketControl (const MIPI_DSI_Type * *base*, uint16_t *wordCount*, uint8_t *virtualChannel*, dsi_tx_data_type_t *dataType*, uint8_t *flags*)

This function configures the next APB packet transfer. After configuration, the packet transfer could be started with function [DSI_SendApbPacket](#). If the packet is long packet, Use [DSI_WriteApbTxPayload](#) to fill the payload before start transfer.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>wordCount</i>	For long packet, this is the byte count of the payload. For short packet, this is (data1 << 8) data0.
<i>virtualChannel</i>	Virtual channel.
<i>dataType</i>	The packet data type, (DI).
<i>flags</i>	The transfer control flags, see <code>_dsi_transfer_flags</code> .

45.2.6.12 void DSI_WriteApbTxPayload (const MIPI_DSI_Type * *base*, const uint8_t * *payload*, uint16_t *payloadSize*)

Write the long packet payload to TX FIFO.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>payload</i>	Pointer to the payload.
<i>payloadSize</i>	Payload size in byte.

45.2.6.13 void DSI_WriteApbTxPayloadExt (const MIPI_DSI_Type * *base*, const uint8_t * *payload*, uint16_t *payloadSize*, bool *sendDscCmd*, uint8_t *dscCmd*)

Write the long packet payload to TX FIFO. This function could be used in two ways

1. Include the DSC command in parameter `payload`. In this case, the DSC command is the first byte of `payload`. The parameter `sendDscCmd` is set to false, the `dscCmd` is not used. This function is the same as [DSI_WriteApbTxPayload](#) when used in this way.

2. The DSC command is not in parameter `payload`, but specified by parameter `dscCmd`. In this case, the parameter `sendDscCmd` is set to true, the `dscCmd` is the DSC command to send. The `payload` is sent after `dscCmd`.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>payload</i>	Pointer to the payload.
<i>payloadSize</i>	Payload size in byte.
<i>sendDscCmd</i>	If set to true, the DSC command is specified by <code>dscCmd</code> , otherwise the DSC command is included in the <code>payload</code> .
<i>dscCmd</i>	The DSC command to send, only used when <code>sendDscCmd</code> is true.

45.2.6.14 **void DSI_ReadApbRxPayload (const MIPI_DSI_Type * *base*, uint8_t * *payload*, uint16_t *payloadSize*)**

Read the long packet payload from RX FIFO. This function reads directly but does not check the RX FIFO status. Upper layer should make sure there are available data.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>payload</i>	Pointer to the payload.
<i>payloadSize</i>	Payload size in byte.

45.2.6.15 **static void DSI_SendApbPacket (const MIPI_DSI_Type * *base*) [inline], [static]**

Send the packet set by [DSI_SetApbPacketControl](#).

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

45.2.6.16 **static uint32_t DSI_GetApbStatus (const MIPI_DSI_Type * *base*) [inline], [static]**

The return value is OR'ed value of `_dsi_apb_status`.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

Returns

The APB status.

**45.2.6.17 static uint32_t DSI_GetRxErrorStatus (const MIPI_DSI_Type * *base*)
[inline], [static]**

The return value is OR'ed value of `_dsi_rx_error_status`.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

Returns

The error status.

**45.2.6.18 static uint8_t DSI_GetEccRxErrorPosition (uint32_t *rxErrorStatus*)
[inline], [static]**

When one-bit ECC RX error detected using [DSI_GetRxErrorStatus](#), this function could be used to get the error bit position.

```
uint8_t eccErrorPos;
uint32_t rxErrorStatus = DSI_GetRxErrorStatus(MIPI_DSI);
if (kDSI_RxErrorEccOneBit & rxErrorStatus)
{
    eccErrorPos = DSI_GetEccRxErrorPosition(rxErrorStatus);
}
```

Parameters

<i>rxErrorStatus</i>	The error status returned by DSI_GetRxErrorStatus .
----------------------	---

Returns

The 1-bit ECC error position.

**45.2.6.19 static uint32_t DSI_GetAndClearHostStatus (const MIPI_DSI_Type * *base*)
[inline], [static]**

The host status are returned as mask value of `_dsi_host_status`.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

Returns

The DSI host status.

45.2.6.20 `static uint32_t DSI_GetRxPacketHeader (const MIPI_DSI_Type * base)
[inline], [static]`

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
-------------	--

Returns

The RX packet header.

45.2.6.21 `static dsi_rx_data_type_t DSI_GetRxPacketType (uint32_t rxPktHeader)
[inline], [static]`

Extract the RX packet type from the packet header get by [DSI_GetRxPacketHeader](#).

Parameters

<i>rxPktHeader</i>	The RX packet header get by DSI_GetRxPacketHeader .
--------------------	---

Returns

The RX packet type.

45.2.6.22 `static uint16_t DSI_GetRxPacketWordCount (uint32_t rxPktHeader)
[inline], [static]`

Extract the RX packet word count from the packet header get by [DSI_GetRxPacketHeader](#).

Parameters

<i>rxPktHeader</i>	The RX packet header get by DSI_GetRxPacketHeader .
--------------------	---

Returns

For long packet, return the payload word count (byte). For short packet, return the $(data0 \ll 8) | data1$.

45.2.6.23 static uint8_t DSI_GetRxPacketVirtualChannel (uint32_t rxPktHeader) [inline], [static]

Extract the RX packet virtual channel from the packet header get by [DSI_GetRxPacketHeader](#).

Parameters

<i>rxPktHeader</i>	The RX packet header get by DSI_GetRxPacketHeader .
--------------------	---

Returns

The virtual channel.

45.2.6.24 status_t DSI_TransferBlocking (const MIPI_DSI_Type * base, dsi_transfer_t * xfer)

Perform APB data transfer using blocking method. This function waits until all data send or received, or timeout happens.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>xfer</i>	Pointer to the transfer structure.

Return values

<i>kStatus_Success</i>	Data transfer finished with no error.
------------------------	---------------------------------------

<i>kStatus_Timeout</i>	Transfer failed because of timeout.
<i>kStatus_DSI_RxData-Error</i>	RX data error, user could use DSI_GetRxErrorStatus to check the error details.
<i>kStatus_DSI_Error-ReportReceived</i>	Error Report packet received, user could use DSI_GetAndClearHostStatus to check the error report status.
<i>kStatus_DSI_Not-Supported</i>	Transfer format not supported.
<i>kStatus_DSI_Fail</i>	Transfer failed for other reasons.

45.2.6.25 **status_t DSI_TransferCreateHandle (const MIPI_DSI_Type * *base*, dsi_handle_t * *handle*, dsi_callback_t *callback*, void * *userData*)**

This function initializes the MIPI DSI handle which can be used for other transactional APIs.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>handle</i>	Handle pointer.
<i>callback</i>	Callback function.
<i>userData</i>	User data.

45.2.6.26 **status_t DSI_TransferNonBlocking (const MIPI_DSI_Type * *base*, dsi_handle_t * *handle*, dsi_transfer_t * *xfer*)**

Perform APB data transfer using interrupt method, when transfer finished, upper layer could be informed through callback function.

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>handle</i>	pointer to dsi_handle_t structure which stores the transfer state.
<i>xfer</i>	Pointer to the transfer structure.

Return values

<i>kStatus_Success</i>	Data transfer started successfully.
<i>kStatus_DSI_Busy</i>	Failed to start transfer because DSI is busy with pervious transfer.
<i>kStatus_DSI_Not-Supported</i>	Transfer format not supported.

45.2.6.27 void DSI_TransferAbort (const MIPI_DSI_Type * *base*, dsi_handle_t * *handle*)

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>handle</i>	pointer to dsi_handle_t structure which stores the transfer state.

45.2.6.28 void DSI_TransferHandleIRQ (const MIPI_DSI_Type * *base*, dsi_handle_t * *handle*)

Parameters

<i>base</i>	MIPI DSI host peripheral base address.
<i>handle</i>	pointer to dsi_handle_t structure which stores the transfer state.

Chapter 46

MU: Messaging Unit

46.1 Overview

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

46.2 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.

46.2.1 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.

46.2.2 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.

46.2.3 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.

46.2.4 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.

46.2.5 MU misc functions

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

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

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

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

Enumerations

- enum `_mu_status_flags` {
`kMU_Tx0EmptyFlag` = (1U << (MU_SR_TEn_SHIFT + 3U)),
`kMU_Tx1EmptyFlag` = (1U << (MU_SR_TEn_SHIFT + 2U)),
`kMU_Tx2EmptyFlag` = (1U << (MU_SR_TEn_SHIFT + 1U)),
`kMU_Tx3EmptyFlag` = (1U << (MU_SR_TEn_SHIFT + 0U)),
`kMU_Rx0FullFlag` = (1U << (MU_SR_RFn_SHIFT + 3U)),
`kMU_Rx1FullFlag` = (1U << (MU_SR_RFn_SHIFT + 2U)),
`kMU_Rx2FullFlag` = (1U << (MU_SR_RFn_SHIFT + 1U)),
`kMU_Rx3FullFlag` = (1U << (MU_SR_RFn_SHIFT + 0U)),
`kMU_GenInt0Flag` = (1U << (MU_SR_GIPn_SHIFT + 3U)),
`kMU_GenInt1Flag` = (1U << (MU_SR_GIPn_SHIFT + 2U)),
`kMU_GenInt2Flag` = (1U << (MU_SR_GIPn_SHIFT + 1U)),
`kMU_GenInt3Flag` = (1U << (MU_SR_GIPn_SHIFT + 0U)),
`kMU_EventPendingFlag` = MU_SR_EP_MASK,
`kMU_FlagsUpdatingFlag` = MU_SR_FUP_MASK,
`kMU_OtherSideInResetFlag` = MU_SR_RS_MASK }
MU status flags.
- enum `_mu_interrupt_enable` {
`kMU_Tx0EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 3U)),
`kMU_Tx1EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 2U)),
`kMU_Tx2EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 1U)),
`kMU_Tx3EmptyInterruptEnable` = (1U << (MU_CR_TIEEn_SHIFT + 0U)),
`kMU_Rx0FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 3U)),
`kMU_Rx1FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 2U)),
`kMU_Rx2FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 1U)),
`kMU_Rx3FullInterruptEnable` = (1U << (MU_CR_RIEEn_SHIFT + 0U)),
`kMU_GenInt0InterruptEnable` = (int)(1U << (MU_CR_GIEEn_SHIFT + 3U)),
`kMU_GenInt1InterruptEnable` = (1U << (MU_CR_GIEEn_SHIFT + 2U)),
`kMU_GenInt2InterruptEnable` = (1U << (MU_CR_GIEEn_SHIFT + 1U)),
`kMU_GenInt3InterruptEnable` = (1U << (MU_CR_GIEEn_SHIFT + 0U)) }
MU interrupt source to enable.
- enum `_mu_interrupt_trigger` {
`kMU_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, 0))
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.

46.3 Macro Definition Documentation

46.3.1 #define FSL_MU_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

46.4 Enumeration Type Documentation

46.4.1 enum _mu_status_flags

Enumerator

kMU_Tx0EmptyFlag TX0 empty.
kMU_Tx1EmptyFlag TX1 empty.
kMU_Tx2EmptyFlag TX2 empty.
kMU_Tx3EmptyFlag TX3 empty.
kMU_Rx0FullFlag RX0 full.
kMU_Rx1FullFlag RX1 full.
kMU_Rx2FullFlag RX2 full.
kMU_Rx3FullFlag RX3 full.
kMU_GenInt0Flag General purpose interrupt 0 pending.
kMU_GenInt1Flag General purpose interrupt 0 pending.
kMU_GenInt2Flag General purpose interrupt 0 pending.
kMU_GenInt3Flag General purpose interrupt 0 pending.
kMU_EventPendingFlag MU event pending.
kMU_FlagsUpdatingFlag MU flags update is on-going.
kMU_OtherSideInResetFlag The other side is in reset.

46.4.2 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.

46.4.3 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.

46.5 Function Documentation

46.5.1 void MU_Init (MU_Type * *base*)

This function enables the MU clock only.

Parameters

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

46.5.2 void MU_Deinit (MU_Type * *base*)

This function disables the MU clock only.

Parameters

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

46.5.3 static void MU_SendMsgNonBlocking (MU_Type * *base*, uint32_t *regIndex*, uint32_t *msg*) [inline], [static]

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

```
* while (!(kMU_Tx0EmptyFlag & MU_GetStatusFlags(base))) { } Wait for TX0
  register empty.
* MU_SendMsgNonBlocking(base, kMU_MsgReg0, MSG_VAL); Write message to the TX0
  register.
*
```

Parameters

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

<i>regIndex</i>	TX register index, see mu_msg_reg_index_t .
<i>msg</i>	Message to send.

46.5.4 void MU_SendMsg (MU_Type * *base*, uint32_t *regIndex*, uint32_t *msg*)

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

Parameters

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	MU message register, see mu_msg_reg_index_t
<i>msg</i>	Message to send.

46.5.5 static uint32_t MU_ReceiveMsgNonBlocking (MU_Type * *base*, uint32_t *regIndex*) [inline], [static]

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

```
* uint32_t msg;
* while (!(kMU_Rx0FullFlag & MU_GetStatusFlags(base)))
* {
* } Wait for the RX0 register full.
*
* msg = MU_ReceiveMsgNonBlocking(base, kMU_MsgReg0); Read message from RX0
* register.
*
```

Parameters

<i>base</i>	MU peripheral base address.
<i>RX</i>	register index, see mu_msg_reg_index_t .

Returns

The received message.

46.5.6 uint32_t MU_ReceiveMsg (MU_Type * *base*, uint32_t *regIndex*)

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

Parameters

<i>base</i>	MU peripheral base address.
<i>regIndex</i>	MU message register, see mu_msg_reg_index_t

Returns

The received message.

46.5.7 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.

46.5.8 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.

46.5.9 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.

46.5.10 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`.

46.5.11 `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.

46.5.12 static void MU_ClearStatusFlags (MU_Type * *base*, uint32_t *mask*) [inline], [static]

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

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

Parameters

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

46.5.13 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> .

46.5.14 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> .

46.5.15 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.

Chapter 47

OCOTP: On Chip One-Time Programmable controller.

47.1 Overview

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

This section contains information describing the requirements for the on-chip eFuse OTP controller along with details about the block functionality and implementation.

47.2 OCOTP function group

The OCOTP driver support operating API to allow read and write the fuse map.

47.2.1 Initialization and de-initialization

The function [OCOTP_Init\(\)](#) is to initialize the OCOTP with peripheral base address and source clock frequency.

The function [OCOTP_Deinit\(\)](#) is to de-initialize the OCOTP controller with peripheral base address.

47.2.2 Read and Write operation

The function [OCOTP_ReloadShadowRegister\(\)](#) is to reload the value from the fuse map. this API should be called firstly before reading the register.

The [OCOTP_ReadFuseShadowRegister\(\)](#) is to read the value from a given address, if operation is success, a known value will be return, othwise, a value of 0xBADABADA will be returned.

The function [OCOTP_WriteFuseShadowRegister\(\)](#) will write a specific value to a known address. please check the return status o make sure whether the access to register is success.

47.3 OCOTP example

This example shows how to get the controller version using API. Due to the eFuse is One-Time programmable, example will only print the information of OCOTP controller version. If more operations are needed, please using the API to implement the write and read operation.

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

Enumerations

- enum {
[kStatus_OCOTP_AccessError](#) = MAKE_STATUS(kStatusGroup_SDK_OCOTP, 0),
[kStatus_OCOTP_CrcFail](#) = MAKE_STATUS(kStatusGroup_SDK_OCOTP, 1),
[kStatus_OCOTP_ReloadError](#),
[kStatus_OCOTP_ProgramFail](#) = MAKE_STATUS(kStatusGroup_SDK_OCOTP, 3),
[kStatus_OCOTP_Locked](#) = MAKE_STATUS(kStatusGroup_SDK_OCOTP, 4) }
_ocotp_status Error codes for the OCOTP driver.

Functions

- void [OCOTP_Init](#) (OCOTP_Type *base, uint32_t srcClock_Hz)
Initializes OCOTP controller.
- void [OCOTP_Deinit](#) (OCOTP_Type *base)
De-initializes OCOTP controller.
- static bool [OCOTP_CheckBusyStatus](#) (OCOTP_Type *base)
Checking the BUSY bit in CTRL register.
- static bool [OCOTP_CheckErrorStatus](#) (OCOTP_Type *base)
Checking the ERROR bit in CTRL register.
- static void [OCOTP_ClearErrorStatus](#) (OCOTP_Type *base)
Clear the error bit if this bit is set.
- status_t [OCOTP_ReloadShadowRegister](#) (OCOTP_Type *base)
Reload the shadow register.
- uint32_t [OCOTP_ReadFuseShadowRegister](#) (OCOTP_Type *base, uint32_t address)
Read the fuse shadow register with the fuse address.
- status_t [OCOTP_ReadFuseShadowRegisterExt](#) (OCOTP_Type *base, uint32_t address, uint32_t *data, uint8_t fuseWords)
Read the fuse shadow register from the fuse address.
- status_t [OCOTP_WriteFuseShadowRegister](#) (OCOTP_Type *base, uint32_t address, uint32_t data)
Write the fuse shadow register with the fuse address and data.
- status_t [OCOTP_WriteFuseShadowRegisterWithLock](#) (OCOTP_Type *base, uint32_t address, uint32_t data, bool lock)
Write the fuse shadow register and lock it.
- static uint32_t [OCOTP_GetVersion](#) (OCOTP_Type *base)
Get the OCOTP controller version from the register.

Driver version

- #define [FSL_OCOTP_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 1, 3))
OCOTP driver version.

47.4 Macro Definition Documentation

47.4.1 #define FSL_OCOTP_DRIVER_VERSION (MAKE_VERSION(2, 1, 3))

47.5 Enumeration Type Documentation

47.5.1 anonymous enum

Enumerator

kStatus_OCOTP_AccessError eFuse and shadow register access error.
kStatus_OCOTP_CrcFail CRC check failed.
kStatus_OCOTP_ReloadError Error happens during reload shadow register.
kStatus_OCOTP_ProgramFail Fuse programming failed.
kStatus_OCOTP_Locked Fuse is locked and cannot be programmed.

47.6 Function Documentation

47.6.1 void OCOTP_Init (OCOTP_Type * *base*, uint32_t *srcClock_Hz*)

Parameters

<i>base</i>	OCOTP peripheral base address.
<i>srcClock_Hz</i>	source clock frequency in unit of Hz. When the macro FSL_FEATURE_OCOTP_HAS_TIMING_CTRL is defined as 0, this parameter is not used, application could pass in 0 in this case.

47.6.2 void OCOTP_Deinit (OCOTP_Type * *base*)

Return values

<i>kStatus_Success</i>	upon successful execution, error status otherwise.
------------------------	--

47.6.3 static bool OCOTP_CheckBusyStatus (OCOTP_Type * *base*) [inline], [static]

Checking this BUSY bit will help confirm if the OCOTP controller is ready for access.

Parameters

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

Return values

<i>true</i>	for bit set and false for cleared.
-------------	------------------------------------

47.6.4 static bool OCOTP_CheckErrorStatus (OCOTP_Type * *base*) [inline], [static]

Parameters

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

Return values

<i>true</i>	for bit set and false for cleared.
-------------	------------------------------------

47.6.5 static void OCOTP_ClearErrorStatus (OCOTP_Type * *base*) [inline], [static]

Parameters

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

47.6.6 status_t OCOTP_ReloadShadowRegister (OCOTP_Type * *base*)

This function will help reload the shadow register without resetting the OCOTP module. Please make sure the OCOTP has been initialized before calling this API.

Parameters

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

Return values

<i>kStatus_Success</i>	Reload success.
------------------------	-----------------

<i>kStatus_OCOTP_Reload-Error</i>	Reload failed.
-----------------------------------	----------------

47.6.7 uint32_t OCOTP_ReadFuseShadowRegister (OCOTP_Type * *base*, uint32_t *address*)

Deprecated Use [OCOTP_ReadFuseShadowRegisterExt](#) instead of this function.

Parameters

<i>base</i>	OCOTP peripheral base address.
<i>address</i>	the fuse address to be read from.

Returns

The read out data.

47.6.8 status_t OCOTP_ReadFuseShadowRegisterExt (OCOTP_Type * *base*, uint32_t *address*, uint32_t * *data*, uint8_t *fuseWords*)

This function reads fuse from *address*, how many words to read is specified by the parameter *fuseWords*. This function could read at most OCOTP_READ_FUSE_DATA_COUNT fuse word one time.

Parameters

<i>base</i>	OCOTP peripheral base address.
<i>address</i>	the fuse address to be read from.
<i>data</i>	Data array to save the readout fuse value.
<i>fuseWords</i>	How many words to read.

Return values

<i>kStatus_Success</i>	Read success.
<i>kStatus_Fail</i>	Error occurs during read.

47.6.9 status_t OCOTP_WriteFuseShadowRegister (OCOTP_Type * *base*, uint32_t *address*, uint32_t *data*)

Please make sure the write address is not locked while calling this API.

Parameters

<i>base</i>	OCOTP peripheral base address.
<i>address</i>	the fuse address to be written.
<i>data</i>	the value will be written to fuse address.

Return values

<i>write</i>	status, kStatus_Success for success and kStatus_Fail for failed.
--------------	--

47.6.10 **status_t OCOTP_WriteFuseShadowRegisterWithLock (OCOTP_Type * *base*, uint32_t *address*, uint32_t *data*, bool *lock*)**

Please make sure the write address is not locked while calling this API.

Some OCOTP controller supports ECC mode and redundancy mode (see reference manual for more details). OCOTP controller will auto select ECC or redundancy mode to program the fuse word according to fuse map definition. In ECC mode, the 32 fuse bits in one word can only be written once. In redundancy mode, the word can be written more than once as long as they are different fuse bits. Set parameter *lock* as true to force use ECC mode.

Parameters

<i>base</i>	OCOTP peripheral base address.
<i>address</i>	The fuse address to be written.
<i>data</i>	The value will be written to fuse address.
<i>lock</i>	Lock or unlock write fuse shadow register operation.

Return values

<i>kStatus_Success</i>	Program and reload success.
<i>kStatus_OCOTP_Locked</i>	The eFuse word is locked and cannot be programmed.
<i>kStatus_OCOTP_ProgramFail</i>	eFuse word programming failed.
<i>kStatus_OCOTP_Reload-Error</i>	eFuse word programming success, but error happens during reload the values.

<i>kStatus_OCOTP_Access-Error</i>	Cannot access eFuse word.
-----------------------------------	---------------------------

47.6.11 static uint32_t OCOTP_GetVersion (OCOTP_Type * *base*) [inline], [static]

Parameters

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

Return values

<i>return</i>	the version value.
---------------	--------------------



Chapter 48

PDM: Microphone Interface

48.1 Overview

Modules

- [PDM Driver](#)
- [PDM EDMA Driver](#)

48.2 PDM Driver

48.2.1 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_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with `kStatus_PDM_Idle` status.

48.2.2 Typical use case

48.2.2.1 PDM receive using an interrupt method

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

48.2.2.2 PDM receive using a SDMA method

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

48.2.2.3 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`

48.2.2.4 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_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.

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) }
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_StatusCh0FifoDataAvaliable` = `PDM_STAT_CH0F_MASK`,
`kPDM_StatusCh1FifoDataAvaliable` = `PDM_STAT_CH1F_MASK`,
`kPDM_StatusCh2FifoDataAvaliable` = `PDM_STAT_CH2F_MASK`,
`kPDM_StatusCh3FifoDataAvaliable` = `PDM_STAT_CH3F_MASK`,
`kPDM_StatusCh4FifoDataAvaliable` = `PDM_STAT_CH4F_MASK`,
`kPDM_StatusCh5FifoDataAvaliable` = `PDM_STAT_CH5F_MASK`,
`kPDM_StatusCh6FifoDataAvaliable` = `PDM_STAT_CH6F_MASK`,
`kPDM_StatusCh7FifoDataAvaliable` = `PDM_STAT_CH7F_MASK` }
The PDM status.
- enum `_pdm_channel_enable_mask` {
`kPDM_EnableChannel0` = `PDM_STAT_CH0F_MASK`,
`kPDM_EnableChannel1` = `PDM_STAT_CH1F_MASK`,
`kPDM_EnableChannel2` = `PDM_STAT_CH2F_MASK`,
`kPDM_EnableChannel3` = `PDM_STAT_CH3F_MASK`,
`kPDM_EnableChannel4` = `PDM_STAT_CH4F_MASK`,
`kPDM_EnableChannel5` = `PDM_STAT_CH5F_MASK`,
`kPDM_EnableChannel6` = `PDM_STAT_CH6F_MASK`,
`kPDM_EnableChannel7` = `PDM_STAT_CH7F_MASK` }
PDM channel enable mask.
- enum `_pdm_fifo_status` {
`kPDM_FifoStatusUnderflowCh0` = `PDM_FIFO_STAT_FIFOUND0_MASK`,
`kPDM_FifoStatusUnderflowCh1` = `PDM_FIFO_STAT_FIFOUND1_MASK`,
`kPDM_FifoStatusUnderflowCh2` = `PDM_FIFO_STAT_FIFOUND2_MASK`,
`kPDM_FifoStatusUnderflowCh3` = `PDM_FIFO_STAT_FIFOUND3_MASK`,
`kPDM_FifoStatusUnderflowCh4` = `PDM_FIFO_STAT_FIFOUND4_MASK`,
`kPDM_FifoStatusUnderflowCh5` = `PDM_FIFO_STAT_FIFOUND5_MASK`,
`kPDM_FifoStatusUnderflowCh6` = `PDM_FIFO_STAT_FIFOUND6_MASK`,
`kPDM_FifoStatusUnderflowCh7` = `PDM_FIFO_STAT_FIFOUND6_MASK`,
`kPDM_FifoStatusOverflowCh0` = `PDM_FIFO_STAT_FIFOOVF0_MASK`,
`kPDM_FifoStatusOverflowCh1` = `PDM_FIFO_STAT_FIFOOVF1_MASK`,
`kPDM_FifoStatusOverflowCh2` = `PDM_FIFO_STAT_FIFOOVF2_MASK`,
`kPDM_FifoStatusOverflowCh3` = `PDM_FIFO_STAT_FIFOOVF3_MASK`,
`kPDM_FifoStatusOverflowCh4` = `PDM_FIFO_STAT_FIFOOVF4_MASK`,
`kPDM_FifoStatusOverflowCh5` = `PDM_FIFO_STAT_FIFOOVF5_MASK`,
`kPDM_FifoStatusOverflowCh6` = `PDM_FIFO_STAT_FIFOOVF6_MASK`,
`kPDM_FifoStatusOverflowCh7` = `PDM_FIFO_STAT_FIFOOVF7_MASK` }
The PDM fifo status.
- enum `_pdm_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_removal_t` {
`kPDM_DcRemoverCutOff21Hz` = 0U,
`kPDM_DcRemoverCutOff83Hz` = 1U,
`kPDM_DcRemoverCutOff152Hz` = 2U,
`kPDM_DcRemoverBypass` = 3U }

PDM DC remover configurations.

- enum `pdm_df_quality_mode_t` {
`kPDM_QualityModeMedium` = 0U,
`kPDM_QualityModeHigh` = 1U,
`kPDM_QualityModeLow` = 7U,
`kPDM_QualityModeVeryLow0` = 6U,
`kPDM_QualityModeVeryLow1` = 5U,
`kPDM_QualityModeVeryLow2` = 4U }

PDM decimation filter quality mode.

- enum `_pdm_qulaity_mode_k_factor` {
`kPDM_QualityModeHighKFactor` = 1U,
`kPDM_QualityModeMediumKFactor` = 2U,
`kPDM_QualityModeLowKFactor` = 4U,
`kPDM_QualityModeVeryLow2KFactor` = 8U }

PDM quality mode K factor.

- enum `pdm_df_output_gain_t` {

```

kPDM_DfOutputGain0 = 0U,
kPDM_DfOutputGain1 = 1U,
kPDM_DfOutputGain2 = 2U,
kPDM_DfOutputGain3 = 3U,
kPDM_DfOutputGain4 = 4U,
kPDM_DfOutputGain5 = 5U,
kPDM_DfOutputGain6 = 6U,
kPDM_DfOutputGain7 = 7U,
kPDM_DfOutputGain8 = 8U,
kPDM_DfOutputGain9 = 9U,
kPDM_DfOutputGain10 = 0xAU,
kPDM_DfOutputGain11 = 0xBU,
kPDM_DfOutputGain12 = 0xCU,
kPDM_DfOutputGain13 = 0xDU,
kPDM_DfOutputGain14 = 0xEU,
kPDM_DfOutputGain15 = 0xFU }

```

PDM decimation filter output gain.

- enum `_pdm_data_width` {
`kPDM_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, 6, 0))
Version 2.6.0.

Initialization and deinitialization

- void [PDM_Init](#) (PDM_Type *base, const [pdm_config_t](#) *config)
Initializes the PDM peripheral.
- void [PDM_Deinit](#) (PDM_Type *base)
De-initializes the PDM peripheral.
- static void [PDM_Reset](#) (PDM_Type *base)
Resets the PDM module.
- static void [PDM_Enable](#) (PDM_Type *base, bool enable)
Enables/disables PDM interface.
- static void [PDM_EnableDoze](#) (PDM_Type *base, bool enable)
Enables/disables DOZE.
- static void [PDM_EnableDebugMode](#) (PDM_Type *base, bool enable)
Enables/disables debug mode for PDM.
- static void [PDM_EnableInDebugMode](#) (PDM_Type *base, bool enable)
Enables/disables PDM interface in debug mode.
- static void [PDM_EnterLowLeakageMode](#) (PDM_Type *base, bool enable)
Enables/disables PDM interface disable/Low Leakage mode.
- static void [PDM_EnableChannel](#) (PDM_Type *base, uint8_t channel, bool enable)
Enables/disables the PDM channel.
- void [PDM_SetChannelConfig](#) (PDM_Type *base, uint32_t channel, const [pdm_channel_config_t](#) *config)
PDM one channel configurations.
- [status_t](#) [PDM_SetSampleRateConfig](#) (PDM_Type *base, uint32_t sourceClock_HZ, uint32_t sampleRate_HZ)
PDM set sample rate.
- [status_t](#) [PDM_SetSampleRate](#) (PDM_Type *base, uint32_t enableChannelMask, [pdm_df_quality_mode_t](#) qualityMode, uint8_t osr, uint32_t clkDiv)
PDM set sample rate.
- uint32_t [PDM_GetInstance](#) (PDM_Type *base)
Get the instance number for PDM.

Status

- static uint32_t [PDM_GetStatus](#) (PDM_Type *base)
Gets the PDM internal status flag.
- static uint32_t [PDM_GetFifoStatus](#) (PDM_Type *base)
Gets the PDM FIFO status flag.
- static uint32_t [PDM_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.

Voice Activity Detector

- void [PDM_SetHwvadConfig](#) (PDM_Type *base, const [pdm_hwvad_config_t](#) *config)
Configure voice activity detector.
- static void [PDM_ForceHwvadOutputDisable](#) (PDM_Type *base, bool enable)
PDM hwvad force output disable.
- static void [PDM_ResetHwvad](#) (PDM_Type *base)
PDM hwvad reset.
- static void [PDM_EnableHwvad](#) (PDM_Type *base, bool enable)
Enable/Disable Voice activity detector.
- static void [PDM_EnableHwvadInterrupts](#) (PDM_Type *base, uint32_t mask)
Enables the PDM Voice Detector interrupt requests.
- static void [PDM_DisableHwvadInterrupts](#) (PDM_Type *base, uint32_t mask)
Disables the PDM Voice Detector interrupt requests.
- static void [PDM_ClearHwvadInterruptStatusFlags](#) (PDM_Type *base, uint32_t mask)
Clears the PDM voice activity detector status flags.
- static uint32_t [PDM_GetHwvadInterruptStatusFlags](#) (PDM_Type *base)
Clears the PDM voice activity detector status flags.
- static uint32_t [PDM_GetHwvadInitialFlag](#) (PDM_Type *base)
Get the PDM voice activity detector initial flags.

- static uint32_t [PDM_GetHwvadVoiceDetectedFlag](#) (PDM_Type *base)
Get the PDM voice activity detector voice detected flags.
- static void [PDM_EnableHwvadSignalFilter](#) (PDM_Type *base, bool enable)
Enables/disables voice activity detector signal filter.
- void [PDM_SetHwvadSignalFilterConfig](#) (PDM_Type *base, bool enableMaxBlock, uint32_t signalGain)
Configure voice activity detector signal filter.
- void [PDM_SetHwvadNoiseFilterConfig](#) (PDM_Type *base, const [pdm_hwvad_noise_filter_t](#) *config)
Configure voice activity detector noise filter.
- static void [PDM_EnableHwvadZeroCrossDetector](#) (PDM_Type *base, bool enable)
Enables/disables voice activity detector zero cross detector.
- void [PDM_SetHwvadZeroCrossDetectorConfig](#) (PDM_Type *base, const [pdm_hwvad_zero_cross_detector_t](#) *config)
Configure voice activity detector zero cross detector.
- static uint16_t [PDM_GetNoiseData](#) (PDM_Type *base)
Reads noise data.
- static void [PDM_SetHwvadInternalFilterStatus](#) (PDM_Type *base, [pdm_hwvad_filter_status_t](#) status)
set hwvad internal filter status .
- void [PDM_SetHwvadInEnvelopeBasedMode](#) (PDM_Type *base, const [pdm_hwvad_config_t](#) *hwvadConfig, const [pdm_hwvad_noise_filter_t](#) *noiseConfig, const [pdm_hwvad_zero_cross_detector_t](#) *zcdConfig, uint32_t signalGain)
set HWVAD in envelope based mode .
- void [PDM_SetHwvadInEnergyBasedMode](#) (PDM_Type *base, const [pdm_hwvad_config_t](#) *hwvadConfig, const [pdm_hwvad_noise_filter_t](#) *noiseConfig, const [pdm_hwvad_zero_cross_detector_t](#) *zcdConfig, uint32_t signalGain)
brief set HWVAD in energy based mode .

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.

48.2.3 Data Structure Documentation

48.2.3.1 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.

48.2.3.2 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.

48.2.3.3 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

48.2.3.4 struct pdm_hwvad_noise_filter_t

Data Fields

- bool [enableAutoNoiseFilter](#)
If noise filterer automatically activated, true means enable.
- bool [enableNoiseMin](#)
If Noise minimum block enabled, true means enabled.
- bool [enableNoiseDecimation](#)
If enable noise input decimation.
- bool [enableNoiseDetectOR](#)
Enables a OR logic in the output of minimum noise estimator block.
- uint32_t [noiseFilterAdjustment](#)
The adjustment value of the noise filter.
- uint32_t [noiseGain](#)
Gain value for the noise energy or envelope estimated.

48.2.3.5 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

48.2.3.6 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

48.2.3.7 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.

48.2.4 Enumeration Type Documentation

48.2.4.1 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.

48.2.4.2 enum _pdm_interrupt_enable

Enumerator

kPDM_ErrorInterruptEnable PDM channel error interrupt enable.
kPDM_FIFOInterruptEnable PDM channel FIFO interrupt.

48.2.4.3 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_StatusCh0FifoDataAvaliable channel 0 fifo data reached watermark level
kPDM_StatusCh1FifoDataAvaliable channel 1 fifo data reached watermark level
kPDM_StatusCh2FifoDataAvaliable channel 2 fifo data reached watermark level
kPDM_StatusCh3FifoDataAvaliable channel 3 fifo data reached watermark level
kPDM_StatusCh4FifoDataAvaliable channel 4 fifo data reached watermark level
kPDM_StatusCh5FifoDataAvaliable channel 5 fifo data reached watermark level
kPDM_StatusCh6FifoDataAvaliable channel 6 fifo data reached watermark level
kPDM_StatusCh7FifoDataAvaliable channel 7 fifo data reached watermark level

48.2.4.4 enum _pdm_channel_enable_mask

Enumerator

kPDM_EnableChannel0 channigel 0 enable mask
kPDM_EnableChannel1 channigel 1 enable mask
kPDM_EnableChannel2 channigel 2 enable mask
kPDM_EnableChannel3 channigel 3 enable mask
kPDM_EnableChannel4 channigel 4 enable mask
kPDM_EnableChannel5 channigel 5 enable mask
kPDM_EnableChannel6 channigel 6 enable mask
kPDM_EnableChannel7 channigel 7 enable mask

48.2.4.5 enum _pdm_fifo_status

Enumerator

kPDM_FifoStatusUnderflowCh0 channel0 fifo status underflow
kPDM_FifoStatusUnderflowCh1 channel1 fifo status underflow
kPDM_FifoStatusUnderflowCh2 channel2 fifo status underflow

kPDM_FifoStatusUnderflowCh3 channel3 fifo status underflow
kPDM_FifoStatusUnderflowCh4 channel4 fifo status underflow
kPDM_FifoStatusUnderflowCh5 channel5 fifo status underflow
kPDM_FifoStatusUnderflowCh6 channel6 fifo status underflow
kPDM_FifoStatusUnderflowCh7 channel7 fifo status underflow
kPDM_FifoStatusOverflowCh0 channel0 fifo status overflow
kPDM_FifoStatusOverflowCh1 channel1 fifo status overflow
kPDM_FifoStatusOverflowCh2 channel2 fifo status overflow
kPDM_FifoStatusOverflowCh3 channel3 fifo status overflow
kPDM_FifoStatusOverflowCh4 channel4 fifo status overflow
kPDM_FifoStatusOverflowCh5 channel5 fifo status overflow
kPDM_FifoStatusOverflowCh6 channel6 fifo status overflow
kPDM_FifoStatusOverflowCh7 channel7 fifo status overflow

48.2.4.6 enum _pdm_range_status

Enumerator

kPDM_RangeStatusUnderFlowCh0 channel0 range status underflow
kPDM_RangeStatusUnderFlowCh1 channel1 range status underflow
kPDM_RangeStatusUnderFlowCh2 channel2 range status underflow
kPDM_RangeStatusUnderFlowCh3 channel3 range status underflow
kPDM_RangeStatusUnderFlowCh4 channel4 range status underflow
kPDM_RangeStatusUnderFlowCh5 channel5 range status underflow
kPDM_RangeStatusUnderFlowCh6 channel6 range status underflow
kPDM_RangeStatusUnderFlowCh7 channel7 range status underflow
kPDM_RangeStatusOverFlowCh0 channel0 range status overflow
kPDM_RangeStatusOverFlowCh1 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

48.2.4.7 enum pdm_dc_removal_t

Enumerator

kPDM_DcRemoverCutOff21Hz DC remover cut off 21HZ.
kPDM_DcRemoverCutOff83Hz DC remover cut off 83HZ.
kPDM_DcRemoverCutOff152Hz DC remover cut off 152HZ.
kPDM_DcRemoverBypass DC remover bypass.

48.2.4.8 enum pdm_df_quality_mode_t

Enumerator

kPDM_QualityModeMedium quality mode medium
kPDM_QualityModeHigh quality mode high
kPDM_QualityModeLow quality mode low
kPDM_QualityModeVeryLow0 quality mode very low0
kPDM_QualityModeVeryLow1 quality mode very low1
kPDM_QualityModeVeryLow2 quality mode very low2

48.2.4.9 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

48.2.4.10 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.

48.2.4.11 enum _pdm_data_width

Enumerator

kPDM_DataWwidth24 PDM data width 24bit.*kPDM_DataWwidth32* PDM data width 32bit.**48.2.4.12 enum _pdm_hwvad_interrupt_enable**

Enumerator

kPDM_HwvadErrorInterruptEnable PDM channel HWVAD error interrupt enable.*kPDM_HwvadInterruptEnable* PDM channel HWVAD interrupt.**48.2.4.13 enum _pdm_hwvad_int_status**

Enumerator

kPDM_HwvadStatusInputSaturation HWVAD saturation condition.*kPDM_HwvadStatusVoiceDetectFlag* HWVAD voice detect interrupt triggered.**48.2.4.14 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.**48.2.4.15 enum pdm_hwvad_filter_status_t**

Enumerator

kPDM_HwvadInternalFilterNormalOperation internal filter ready for normal operation*kPDM_HwvadInternalFilterInitial* interla filter are initial**48.2.4.16 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

48.2.5 Function Documentation

48.2.5.1 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.

48.2.5.2 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
-------------	------------------

48.2.5.3 static void PDM_Reset (PDM_Type * *base*) [inline], [static]

Parameters

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

48.2.5.4 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.

48.2.5.5 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.

48.2.5.6 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.

48.2.5.7 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.

48.2.5.8 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.

48.2.5.9 `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.

48.2.5.10 `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.

48.2.5.11 `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.

48.2.5.12 `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 superseded 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

48.2.5.13 `uint32_t PDM_GetInstance (PDM_Type * base)`

Parameters

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

48.2.5.14 `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.

48.2.5.15 **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.

48.2.5.16 **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.

48.2.5.17 **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_StatusCh7FifoDataAvaliable.

48.2.5.18 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.

48.2.5.19 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.

48.2.5.20 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"> • kPDM_ErrorInterruptEnable • kPDM_FIFOInterruptEnable

48.2.5.21 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"> • kPDM_ErrorInterruptEnable • kPDM_FIFOInterruptEnable

48.2.5.22 `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.

48.2.5.23 `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.

48.2.5.24 `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.

48.2.5.25 `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.

48.2.5.26 `void PDM_SetHwvadConfig (PDM_Type * base, const pdm_hwvad_config_t * config)`

Parameters

<i>base</i>	PDM base pointer
<i>config</i>	Voice activity detector configure structure pointer .

48.2.5.27 `static void PDM_ForceHwvadOutputDisable (PDM_Type * base, bool enable)`
[inline], [static]

Parameters

<i>base</i>	PDM base pointer
<i>enable</i>	true is output force disable, false is output not force.

48.2.5.28 static void PDM_ResetHwvad (PDM_Type * *base*) [inline], [static]

It will reset VADNDATA register and will clean all internal buffers, should be called when the PDM isn't running.

Parameters

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

48.2.5.29 static void PDM_EnableHwvad (PDM_Type * *base*, bool *enable*) [inline], [static]

Should be called when the PDM isn't running.

Parameters

<i>base</i>	PDM base pointer.
<i>enable</i>	True means enable voice activity detector, false means disable.

48.2.5.30 static void PDM_EnableHwvadInterrupts (PDM_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	PDM base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kPDM_HWVADErrorInterruptEnable • kPDM_HWVADInterruptEnable

48.2.5.31 static void PDM_DisableHwvadInterrupts (PDM_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	PDM base pointer
<i>mask</i>	interrupt source The parameter can be a combination of the following sources if defined. <ul style="list-style-type: none"> • kPDM_HWVADErrorInterruptEnable • kPDM_HWVADInterruptEnable

48.2.5.32 `static void PDM_ClearHwvadInterruptStatusFlags (PDM_Type * base, uint32_t mask) [inline], [static]`

Parameters

<i>base</i>	PDM base pointer
<i>mask</i>	State mask,reference _pdm_hwvad_int_status.

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

Parameters

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

Returns

status, reference _pdm_hwvad_int_status

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

Parameters

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

Returns

initial flag.

48.2.5.35 `static uint32_t PDM_GetHwvadVoiceDetectedFlag (PDM_Type * base)`
`[inline], [static]`

NOte: this flag is auto cleared when voice gone.

Parameters

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

Returns

voice detected flag.

48.2.5.36 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.

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

Parameters

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

48.2.5.38 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 .

48.2.5.39 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.

48.2.5.40 `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 .

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

Parameters

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

Returns

Data in PDM noise data register.

48.2.5.42 `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.

48.2.5.43 `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.

48.2.5.44 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.

48.2.5.45 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.

48.2.5.46 `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.
--	---------------------

48.2.5.47 `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 <code>pdm_handle_t</code> 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.

48.2.5.48 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 <code>pdm_handle_t</code> structure which stores the transfer state.

48.2.5.49 void PDM_TransferHandleIRQ (PDM_Type * *base*, `pdm_handle_t` * *handle*)

Parameters

<i>base</i>	PDM base pointer.
<i>handle</i>	Pointer to the <code>pdm_handle_t</code> structure.

48.3 PDM EDMA Driver

48.3.1 Overview

Data Structures

- struct [pdm_edma_transfer_t](#)
PDM edma transfer. [More...](#)
- struct [pdm_edma_handle_t](#)
PDM DMA transfer handle, users should not touch the content of the handle. [More...](#)

Typedefs

- typedef void(* [pdm_edma_callback_t](#))(PDM_Type *base, pdm_edma_handle_t *handle, [status_t](#) status, void *userData)
PDM eDMA transfer callback function for finish and error.

Driver version

- #define [FSL_PDM_EDMA_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 5, 0))
Version 2.5.0.

PDM eDMA Transactional

- void [PDM_TransferInstallEDMATCDMemory](#) (pdm_edma_handle_t *handle, void *tcdAddr, size_t tcdNum)
Install EDMA descriptor memory.
- void [PDM_TransferCreateHandleEDMA](#) (PDM_Type *base, pdm_edma_handle_t *handle, [pdm_edma_callback_t](#) callback, void *userData, [edma_handle_t](#) *dmaHandle)
Initializes the PDM Rx eDMA handle.
- void [PDM_TransferSetChannelConfigEDMA](#) (PDM_Type *base, pdm_edma_handle_t *handle, uint32_t channel, const [pdm_channel_config_t](#) *config)
Configures the PDM channel.
- [status_t](#) [PDM_TransferReceiveEDMA](#) (PDM_Type *base, pdm_edma_handle_t *handle, [pdm_edma_transfer_t](#) *xfer)
Performs a non-blocking PDM receive using eDMA.
- void [PDM_TransferTerminateReceiveEDMA](#) (PDM_Type *base, pdm_edma_handle_t *handle)
Terminate all PDM receive.
- void [PDM_TransferAbortReceiveEDMA](#) (PDM_Type *base, pdm_edma_handle_t *handle)
Aborts a PDM receive using eDMA.
- [status_t](#) [PDM_TransferGetReceiveCountEDMA](#) (PDM_Type *base, pdm_edma_handle_t *handle, size_t *count)
Gets byte count received by PDM.

48.3.2 Data Structure Documentation

48.3.2.1 struct pdm_edma_transfer_t

Data Fields

- volatile uint8_t * [data](#)
Data start address to transfer.
- volatile size_t [dataSize](#)
Total Transfer bytes size.
- struct _pdm_edma_transfer * [linkTransfer](#)
linked transfer configurations

Field Documentation

(1) volatile uint8_t* pdm_edma_transfer_t::data

(2) volatile size_t pdm_edma_transfer_t::dataSize

48.3.2.2 struct _pdm_edma_handle

PDM edma handler.

Data Fields

- [edma_handle_t](#) * [dmaHandle](#)
DMA handler for PDM send.
- uint8_t [count](#)
The transfer data count in a DMA request.
- uint32_t [receivedBytes](#)
total transfer count
- uint32_t [state](#)
Internal state for PDM eDMA transfer.
- [pdm_edma_callback_t](#) [callback](#)
Callback for users while transfer finish or error occurs.
- bool [isLoopTransfer](#)
loop transfer
- void * [userData](#)
User callback parameter.
- [edma_tcd_t](#) * [tcd](#)
TCD pool for eDMA transfer.
- uint32_t [tcdNum](#)
TCD number.
- uint32_t [tcdUser](#)
Index for user to queue transfer.
- uint32_t [tcdDriver](#)
Index for driver to get the transfer data and size.
- volatile uint32_t [tcdUsedNum](#)
Index for user to queue transfer.

- uint8_t [endChannel](#)
The last enabled channel.
- uint8_t [channelNums](#)
total channel numbers

Field Documentation

- (1) `edma_tcd_t* pdm_edma_handle_t::tcd`
- (2) `uint32_t pdm_edma_handle_t::tcdUser`
- (3) `volatile uint32_t pdm_edma_handle_t::tcdUsedNum`

48.3.3 Function Documentation

48.3.3.1 `void PDM_TransferInstalledEDMATCDMemory (pdm_edma_handle_t * handle, void * tcdAddr, size_t tcdNum)`

Parameters

<i>handle</i>	Pointer to EDMA channel transfer handle.
<i>tcdAddr</i>	EDMA head descriptor address.
<i>tcdNum</i>	EDMA link descriptor address.

48.3.3.2 `void PDM_TransferCreateHandleEDMA (PDM_Type * base, pdm_edma_handle_t * handle, pdm_edma_callback_t callback, void * userData, edma_handle_t * dmaHandle)`

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

Parameters

<i>base</i>	PDM base pointer.
<i>handle</i>	PDM eDMA handle pointer.
<i>base</i>	PDM peripheral base address.
<i>callback</i>	Pointer to user callback function.

<i>userData</i>	User parameter passed to the callback function.
<i>dmaHandle</i>	eDMA handle pointer, this handle shall be static allocated by users.

**48.3.3.3 void PDM_TransferSetChannelConfigEDMA (PDM_Type * *base*,
pdm_edma_handle_t * *handle*, uint32_t *channel*, const pdm_channel_config_t *
config)**

Parameters

<i>base</i>	PDM base pointer.
<i>handle</i>	PDM eDMA handle pointer.
<i>channel</i>	channel index.
<i>config</i>	pdm channel configurations.

**48.3.3.4 status_t PDM_TransferReceiveEDMA (PDM_Type * *base*, pdm_edma_handle_t *
handle, pdm_edma_transfer_t * *xfer*)**

Note

This interface returns immediately after the transfer initiates. Call the PDM_GetReceiveRemaining-Bytes to poll the transfer status and check whether the PDM transfer is finished.

1. Scatter gather case: This function support dynamic scatter gather and static scatter gather, a. for the dynamic scatter gather case: Application should call PDM_TransferReceiveEDMA function continuously to make sure new receive request is submit before the previous one finish. b. for the static scatter gather case: Application should use the link transfer feature and make sure a loop link transfer is provided, such as:

```
* pdm_edma_transfer_t pdmXfer[2] =
* {
*     {
*         .data = s_buffer,
*         .dataSize = BUFFER_SIZE,
*         .linkTransfer = &pdmXfer[1],
*     },
*     {
*         .data = &s_buffer[BUFFER_SIZE],
*         .dataSize = BUFFER_SIZE,
*         .linkTransfer = &pdmXfer[0]
*     },
* };
*
```

2. Multi channel case: This function support receive multi pdm channel data, for example, if two channel is requested,

```

* PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0,
  DEMO_PDM_ENABLE_CHANNEL_0, &channelConfig);
* PDM_TransferSetChannelConfigEDMA(DEMO_PDM, &s_pdmRxHandle_0,
  DEMO_PDM_ENABLE_CHANNEL_1, &channelConfig);
* PDM_TransferReceiveEDMA(DEMO_PDM, &s_pdmRxHandle_0, pdmXfer);
*

```

Then the output data will be formatted as:

|CHANNEL0 | CHANNEL1 | CHANNEL0 | CHANNEL1 | CHANNEL0 | CHANNEL 1 ||

Parameters

<i>base</i>	PDM base pointer
<i>handle</i>	PDM eDMA handle pointer.
<i>xfer</i>	Pointer to DMA transfer structure.

Return values

<i>kStatus_Success</i>	Start a PDM eDMA receive successfully.
<i>kStatus_InvalidArgument</i>	The input argument is invalid.
<i>kStatus_RxBusy</i>	PDM is busy receiving data.

48.3.3.5 void PDM_TransferTerminateReceiveEDMA (PDM_Type * *base*, pdm_edma_handle_t * *handle*)

This function will clear all transfer slots buffered in the pdm queue. If users only want to abort the current transfer slot, please call PDM_TransferAbortReceiveEDMA.

Parameters

<i>base</i>	PDM base pointer.
<i>handle</i>	PDM eDMA handle pointer.

48.3.3.6 void PDM_TransferAbortReceiveEDMA (PDM_Type * *base*, pdm_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 PDM_TransferTerminateReceiveEDMA.

Parameters

<i>base</i>	PDM base pointer
<i>handle</i>	PDM eDMA handle pointer.

48.3.3.7 **status_t PDM_TransferGetReceiveCountEDMA (PDM_Type * *base*, pdm_edma_handle_t * *handle*, size_t * *count*)**

Parameters

<i>base</i>	PDM base pointer
<i>handle</i>	PDM eDMA handle pointer.
<i>count</i>	Bytes count received by PDM.

Return values

<i>kStatus_Success</i>	Succeed get the transfer count.
<i>kStatus_NoTransferInProgress</i>	There is no non-blocking transaction in progress.

Chapter 49

PIT: Periodic Interrupt Timer

49.1 Overview

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

49.2 Function groups

The PIT driver supports operating the module as a time counter.

49.2.1 Initialization and deinitialization

The function [PIT_Init\(\)](#) initializes the PIT with specified configurations. The function [PIT_GetDefaultConfig\(\)](#) gets the default configurations. The initialization function configures the PIT operation in debug mode.

The function [PIT_SetTimerChainMode\(\)](#) configures the chain mode operation of each PIT channel.

The function [PIT_Deinit\(\)](#) disables the PIT timers and disables the module clock.

49.2.2 Timer period Operations

The function [PITR_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 [PIT_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. Users can call the utility macros provided in `fsl_common.h` to convert to microseconds or milliseconds.

49.2.3 Start and Stop timer operations

The function [PIT_StartTimer\(\)](#) starts the timer counting. After calling this function, the timer loads the period value set earlier via the [PIT_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 [PIT_StopTimer\(\)](#) stops the timer counting.

49.2.4 Status

Provides functions to get and clear the PIT status.

49.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

49.3 Typical use case

49.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pit`

Data Structures

- struct `pit_config_t`
PIT configuration structure. [More...](#)

Enumerations

- enum `pit_chnl_t` {
 `kPIT_Chnl_0` = 0U,
 `kPIT_Chnl_1`,
 `kPIT_Chnl_2`,
 `kPIT_Chnl_3` }
List of PIT channels.
- enum `pit_interrupt_enable_t` { `kPIT_TimerInterruptEnable` = `PIT_TCTRL_TIE_MASK` }
List of PIT interrupts.
- enum `pit_status_flags_t` { `kPIT_TimerFlag` = `PIT_TFLG_TIF_MASK` }
List of PIT status flags.

Functions

- `uint64_t PIT_GetLifetimeTimerCount` (`PIT_Type *base`)
Reads the current lifetime counter value.

Driver version

- `#define FSL_PIT_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 4)`)
PIT Driver Version 2.0.4.

Initialization and deinitialization

- `void PIT_Init` (`PIT_Type *base`, `const pit_config_t *config`)

- *Ungates the PIT clock, enables the PIT module, and configures the peripheral for basic operations.*
- void [PIT_Deinit](#) (PIT_Type *base)
- *Gates the PIT clock and disables the PIT module.*
- static void [PIT_GetDefaultConfig](#) ([pit_config_t](#) *config)
- *Fills in the PIT configuration structure with the default settings.*
- static void [PIT_SetTimerChainMode](#) (PIT_Type *base, [pit_chnl_t](#) channel, bool enable)
- *Enables or disables chaining a timer with the previous timer.*

Interrupt Interface

- static void [PIT_EnableInterrupts](#) (PIT_Type *base, [pit_chnl_t](#) channel, uint32_t mask)
- *Enables the selected PIT interrupts.*
- static void [PIT_DisableInterrupts](#) (PIT_Type *base, [pit_chnl_t](#) channel, uint32_t mask)
- *Disables the selected PIT interrupts.*
- static uint32_t [PIT_GetEnabledInterrupts](#) (PIT_Type *base, [pit_chnl_t](#) channel)
- *Gets the enabled PIT interrupts.*

Status Interface

- static uint32_t [PIT_GetStatusFlags](#) (PIT_Type *base, [pit_chnl_t](#) channel)
- *Gets the PIT status flags.*
- static void [PIT_ClearStatusFlags](#) (PIT_Type *base, [pit_chnl_t](#) channel, uint32_t mask)
- *Clears the PIT status flags.*

Read and Write the timer period

- static void [PIT_SetTimerPeriod](#) (PIT_Type *base, [pit_chnl_t](#) channel, uint32_t count)
- *Sets the timer period in units of count.*
- static uint32_t [PIT_GetCurrentTimerCount](#) (PIT_Type *base, [pit_chnl_t](#) channel)
- *Reads the current timer counting value.*

Timer Start and Stop

- static void [PIT_StartTimer](#) (PIT_Type *base, [pit_chnl_t](#) channel)
- *Starts the timer counting.*
- static void [PIT_StopTimer](#) (PIT_Type *base, [pit_chnl_t](#) channel)
- *Stops the timer counting.*

49.4 Data Structure Documentation

49.4.1 struct pit_config_t

This structure holds the configuration settings for the PIT peripheral. To initialize this structure to reasonable defaults, call the [PIT_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The configuration structure can be made constant so it resides in flash.

Data Fields

- bool [enableRunInDebug](#)
true: Timers run in debug mode; false: Timers stop in debug mode

49.5 Enumeration Type Documentation

49.5.1 enum pit_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

kPIT_Chnl_0 PIT channel number 0.
kPIT_Chnl_1 PIT channel number 1.
kPIT_Chnl_2 PIT channel number 2.
kPIT_Chnl_3 PIT channel number 3.

49.5.2 enum pit_interrupt_enable_t

Enumerator

kPIT_TimerInterruptEnable Timer interrupt enable.

49.5.3 enum pit_status_flags_t

Enumerator

kPIT_TimerFlag Timer flag.

49.6 Function Documentation

49.6.1 void PIT_Init (PIT_Type * *base*, const pit_config_t * *config*)

Note

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

Parameters

<i>base</i>	PIT peripheral base address
<i>config</i>	Pointer to the user's PIT config structure

49.6.2 void PIT_Deinit (PIT_Type * *base*)

Parameters

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

49.6.3 static void PIT_GetDefaultConfig (pit_config_t * *config*) [inline], [static]

The default values are as follows.

```
* config->enableRunInDebug = false;
*
```

Parameters

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

49.6.4 static void PIT_SetTimerChainMode (PIT_Type * *base*, pit_chnl_t *channel*, bool *enable*) [inline], [static]

When a timer has a chain mode enabled, it only counts after the previous timer has expired. If the timer n-1 has counted down to 0, counter n decrements the value by one. Each timer is 32-bits, which allows the developers to chain timers together and form a longer timer (64-bits and larger). The first timer (timer 0) can't be chained to any other timer.

Parameters

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

<i>channel</i>	Timer channel number which is chained with the previous timer
<i>enable</i>	Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers.

49.6.5 static void PIT_EnableInterrupts (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The interrupts to enable. This is a logical OR of members of the enumeration pit_interrupt_enable_t

49.6.6 static void PIT_DisableInterrupts (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The interrupts to disable. This is a logical OR of members of the enumeration pit_interrupt_enable_t

49.6.7 static uint32_t PIT_GetEnabledInterrupts (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration [pit_interrupt_enable_t](#)

49.6.8 `static uint32_t PIT_GetStatusFlags (PIT_Type * base, pit_chnl_t channel)`
`[inline], [static]`

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration [pit_status_flags_t](#)

49.6.9 static void PIT_ClearStatusFlags (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number
<i>mask</i>	The status flags to clear. This is a logical OR of members of the enumeration pit_status_flags_t

49.6.10 static void PIT_SetTimerPeriod (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *count*) [inline], [static]

Timers begin counting from the value set by this function until it reaches 0, then it generates an interrupt and load 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

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

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number

<i>count</i>	Timer period in units of ticks
--------------	--------------------------------

49.6.11 **static uint32_t PIT_GetCurrentTimerCount (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]**

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Note

Users can call the utility macros provided in fsl_common.h to convert ticks to usec or msec.

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number

Returns

Current timer counting value in ticks

49.6.12 **static void PIT_StartTimer (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]**

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number.

49.6.13 **static void PIT_StopTimer (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]**

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

<i>base</i>	PIT peripheral base address
<i>channel</i>	Timer channel number.

49.6.14 uint64_t PIT_GetLifetimeTimerCount (PIT_Type * *base*)

The lifetime timer is a 64-bit timer which chains timer 0 and timer 1 together. Timer 0 and 1 are chained by calling the PIT_SetTimerChainMode before using this timer. The period of lifetime timer is equal to the "period of timer 0 * period of timer 1". For the 64-bit value, the higher 32-bit has the value of timer 1, and the lower 32-bit has the value of timer 0.

Parameters

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

Returns

Current lifetime timer value

Chapter 50

PUF: Physical Unclonable Function

50.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Physical Unclonable Function (PUF) module of MCUXpresso SDK devices. The PUF controller provides a secure key storage without injecting or provisioning device unique PUF root key.

Blocking synchronous APIs are provided for generating the activation code, intrinsic key generation, storing and reconstructing keys using PUF hardware. The PUF operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until an PUF operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

50.2 PUF Driver Initialization and deinitialization

PUF Driver is initialized by calling the PUF_Init() function, it resets the PUF module, enables its clock and enables power to PUF SRAM. PUF Driver is deinitialized by calling the PUF_Deinit() function, it disables PUF module clock, asserts peripheral reset and disables power to PUF SRAM.

50.3 Comments about API usage in RTOS

PUF operations provided by this driver are not re-entrant. Thus, application software shall ensure the PUF module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

50.4 Comments about API usage in interrupt handler

All APIs can be used from interrupt handler although execution time shall be considered (interrupt latency of equal and lower priority interrupts increases).

50.5 PUF Driver Examples

50.5.1 Simple examples

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

Macros

- #define **PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE**(x) (((160u + (((x) << 3) + 255u) >> 8) << 8)) >> 3)
Get Key Code size in bytes from key size in bytes at compile time.

Enumerations

- enum `puf_key_slot_t` {
`kPUF_KeySlot0` = 0U,
`kPUF_KeySlot1` = 1U }
PUF key slot.
- enum
PUF status return codes.

Driver version

- #define `FSL_PUF_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 6)`)
PUF driver version.

50.6 Macro Definition Documentation

50.6.1 #define FSL_PUF_DRIVER_VERSION (MAKE_VERSION(2, 1, 6))

Version 2.1.6.

Current version: 2.1.6

Change log:

- 2.0.0
 - Initial version.
- 2.0.1
 - Fixed `puf_wait_usec` function optimization issue.
- 2.0.2
 - Add PUF configuration structure and support for PUF SRAM controller. Remove magic constants.
- 2.0.3
 - Fix MISRA C-2012 issue.
- 2.1.0
 - Align driver with PUF SRAM controller registers on LPCXpresso55s16.
 - Update initialization logic .
- 2.1.1
 - Fix ARMGCC build warning .
- 2.1.2
 - Update: Add automatic big to little endian swap for user (pre-shared) keys destined to secret hardware bus (PUF key index 0).
- 2.1.3
 - Fix MISRA C-2012 issue.
- 2.1.4
 - Replace register `uint32_t ticksCount` with volatile `uint32_t ticksCount` in `puf_wait_usec()` to prevent optimization out delay loop.
- 2.1.5
 - Use common SDK delay in `puf_wait_usec()`

- 2.1.6
 - Changed wait time in PUF_Init(), when initialization fails it will try PUF_Powercycle() with shorter time. If this shorter time will also fail, initialization will be tried with worst case time as before.

50.6.2 `#define PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(x) ((160u + (((x) << 3) + 255u) >> 8) << 8)) >> 3)`

50.7 Enumeration Type Documentation

50.7.1 enum puf_key_slot_t

Enumerator

kPUF_KeySlot0 PUF key slot 0.
kPUF_KeySlot1 PUF key slot 1.

50.7.2 anonymous enum

Chapter 51

PWM: Pulse Width Modulator

51.1 Overview

The MCUXpresso SDK provides a driver for the Pulse Width Modulator (PWM) of MCUXpresso SDK devices.

51.2 PWM: Pulse Width Modulator

51.2.1 Initialization and deinitialization

The function [PWM_Init\(\)](#) initializes the PWM sub module with specified configurations, the function [PWM_GetDefaultConfig\(\)](#) could help to get the default configurations. The initialization function configures the sub module for the requested register update mode for registers with buffers. It also sets up the sub module operation in debug and wait modes.

51.2.2 PWM Operations

The function [PWM_SetupPwm\(\)](#) sets up PWM channels for PWM output, the function can set up PWM signal properties for multiple channels. The PWM has 2 channels: A and B. Each channel has its own duty cycle and level-mode specified, however the same PWM period and PWM mode is applied to all channels requesting PWM output. The signal duty cycle is provided as a percentage of the PWM period, its value should be between 0 and 100; 0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle). The function also sets up the channel dead time value which is used when the user selects complementary mode of operation.

The function [PWM_UpdatePwmDutycycle\(\)](#) updates the PWM signal duty cycle of a particular PWM channel.

51.2.3 Input capture operations

The function [PWM_SetupInputCapture\(\)](#) sets up a PWM channel for input capture. The user can specify the capture edge and the mode; one-shot capture or free-running capture.

51.2.4 Fault operation

The function [PWM_SetupFault\(\)](#) sets up the properties for each fault.

51.2.5 PWM Start and Stop operations

The function [PWM_StartTimer\(\)](#) can be used to start one or multiple sub modules. The function [PWM_StopTimer\(\)](#) can be used to stop one or multiple sub modules.

51.2.6 Status

Provide functions to get and clear the PWM status.

51.2.7 Interrupt

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

51.3 Register Update

Some of the PWM registers have buffers, the driver support various methods to update these registers with the content of the register buffer. The update mechanism for register with buffers can be specified through the following fields available in the configuration structure. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm` The user can select one of the reload options provided in enumeration [pwm_register_reload_t](#). When using immediate reload, the `reloadFrequency` field is not used.

The driver initialization function sets up the appropriate bits in the PWM module based on the register update options selected.

The below function should be used to initiate a register reload. The example shows register reload initiated on PWM sub modules 0, 1, and 2. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm`

51.4 Typical use case

51.4.1 PWM output

Output PWM signal on 3 PWM sub module with different dutycycles. Periodically update the PWM signal duty cycle. Each sub module runs in Complementary output mode with PWM A used to generate the complementary PWM pair. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pwm`

Data Structures

- struct [pwm_signal_param_t](#)
Structure for the user to define the PWM signal characteristics. [More...](#)
- struct [pwm_config_t](#)
PWM config structure. [More...](#)
- struct [pwm_fault_input_filter_param_t](#)

- *Structure for the user to configure the fault input filter. [More...](#)*
- struct `pwm_fault_param_t`
Structure is used to hold the parameters to configure a PWM fault. [More...](#)
- struct `pwm_input_capture_param_t`
Structure is used to hold parameters to configure the capture capability of a signal pin. [More...](#)

Macros

- #define `PWM_SUBMODULE_SWCONTROL_WIDTH` 2
Number of bits per submodule for software output control.

Enumerations

- enum `pwm_submodule_t` {
 `kPWM_Module_0` = 0U,
 `kPWM_Module_1`,
 `kPWM_Module_2`,
 `kPWM_Module_3` }
List of PWM submodules.
- enum `pwm_channels_t`
List of PWM channels in each module.
- enum `pwm_value_register_t` {
 `kPWM_ValueRegister_0` = 0U,
 `kPWM_ValueRegister_1`,
 `kPWM_ValueRegister_2`,
 `kPWM_ValueRegister_3`,
 `kPWM_ValueRegister_4`,
 `kPWM_ValueRegister_5` }
List of PWM value registers.
- enum `_pwm_value_register_mask` {
 `kPWM_ValueRegisterMask_0` = (1U << 0),
 `kPWM_ValueRegisterMask_1` = (1U << 1),
 `kPWM_ValueRegisterMask_2` = (1U << 2),
 `kPWM_ValueRegisterMask_3` = (1U << 3),
 `kPWM_ValueRegisterMask_4` = (1U << 4),
 `kPWM_ValueRegisterMask_5` = (1U << 5) }
List of PWM value registers mask.
- enum `pwm_clock_source_t` {
 `kPWM_BusClock` = 0U,
 `kPWM_ExternalClock`,
 `kPWM_Submodule0Clock` }
PWM clock source selection.
- enum `pwm_clock_prescale_t` {

- ```

kPWM_Prescale_Divide_1 = 0U,
kPWM_Prescale_Divide_2,
kPWM_Prescale_Divide_4,
kPWM_Prescale_Divide_8,
kPWM_Prescale_Divide_16,
kPWM_Prescale_Divide_32,
kPWM_Prescale_Divide_64,
kPWM_Prescale_Divide_128 }

```
- PWM prescaler factor selection for clock source.*
- enum `pwm_force_output_trigger_t` {

```

kPWM_Force_Local = 0U,
kPWM_Force_Master,
kPWM_Force_LocalReload,
kPWM_Force_MasterReload,
kPWM_Force_LocalSync,
kPWM_Force_MasterSync,
kPWM_Force_External,
kPWM_Force_ExternalSync }

```

*Options that can trigger a PWM FORCE\_OUT.*
  - enum `pwm_init_source_t` {

```

kPWM_Initialize_LocalSync = 0U,
kPWM_Initialize_MasterReload,
kPWM_Initialize_MasterSync,
kPWM_Initialize_ExtSync }

```

*PWM counter initialization options.*
  - enum `pwm_load_frequency_t` {

```

kPWM_LoadEveryOportunity = 0U,
kPWM_LoadEvery2Oportunity,
kPWM_LoadEvery3Oportunity,
kPWM_LoadEvery4Oportunity,
kPWM_LoadEvery5Oportunity,
kPWM_LoadEvery6Oportunity,
kPWM_LoadEvery7Oportunity,
kPWM_LoadEvery8Oportunity,
kPWM_LoadEvery9Oportunity,
kPWM_LoadEvery10Oportunity,
kPWM_LoadEvery11Oportunity,
kPWM_LoadEvery12Oportunity,
kPWM_LoadEvery13Oportunity,
kPWM_LoadEvery14Oportunity,
kPWM_LoadEvery15Oportunity,
kPWM_LoadEvery16Oportunity }

```

*PWM load frequency selection.*
  - enum `pwm_fault_input_t` {

```
kPWM_Fault_0 = 0U,
kPWM_Fault_1,
kPWM_Fault_2,
kPWM_Fault_3 }
```

*List of PWM fault selections.*

- enum `pwm_fault_disable_t` {  
`kPWM_FaultDisable_0` = (1U << 0),  
`kPWM_FaultDisable_1` = (1U << 1),  
`kPWM_FaultDisable_2` = (1U << 2),  
`kPWM_FaultDisable_3` = (1U << 3) }

*List of PWM fault disable mapping selections.*

- enum `pwm_fault_channels_t`

*List of PWM fault channels.*

- enum `pwm_input_capture_edge_t` {  
`kPWM_Disable` = 0U,  
`kPWM_FallingEdge`,  
`kPWM_RisingEdge`,  
`kPWM_RiseAndFallEdge` }

*PWM capture edge select.*

- enum `pwm_force_signal_t` {  
`kPWM_UsePwm` = 0U,  
`kPWM_InvertedPwm`,  
`kPWM_SoftwareControl`,  
`kPWM_UseExternal` }

*PWM output options when a FORCE\_OUT signal is asserted.*

- enum `pwm_chnl_pair_operation_t` {  
`kPWM_Independent` = 0U,  
`kPWM_ComplementaryPwmA`,  
`kPWM_ComplementaryPwmB` }

*Options available for the PWM A & B pair operation.*

- enum `pwm_register_reload_t` {  
`kPWM_ReloadImmediate` = 0U,  
`kPWM_ReloadPwmHalfCycle`,  
`kPWM_ReloadPwmFullCycle`,  
`kPWM_ReloadPwmHalfAndFullCycle` }

*Options available on how to load the buffered-registers with new values.*

- enum `pwm_fault_recovery_mode_t` {  
`kPWM_NoRecovery` = 0U,  
`kPWM_RecoverHalfCycle`,  
`kPWM_RecoverFullCycle`,  
`kPWM_RecoverHalfAndFullCycle` }

*Options available on how to re-enable the PWM output when recovering from a fault.*

- enum `pwm_interrupt_enable_t` {

```

kPWM_CompareVal0InterruptEnable = (1U << 0),
kPWM_CompareVal1InterruptEnable = (1U << 1),
kPWM_CompareVal2InterruptEnable = (1U << 2),
kPWM_CompareVal3InterruptEnable = (1U << 3),
kPWM_CompareVal4InterruptEnable = (1U << 4),
kPWM_CompareVal5InterruptEnable = (1U << 5),
kPWM_CaptureX0InterruptEnable = (1U << 6),
kPWM_CaptureX1InterruptEnable = (1U << 7),
kPWM_CaptureB0InterruptEnable = (1U << 8),
kPWM_CaptureB1InterruptEnable = (1U << 9),
kPWM_CaptureA0InterruptEnable = (1U << 10),
kPWM_CaptureA1InterruptEnable = (1U << 11),
kPWM_ReloadInterruptEnable = (1U << 12),
kPWM_ReloadErrorInterruptEnable = (1U << 13),
kPWM_Fault0InterruptEnable = (1U << 16),
kPWM_Fault1InterruptEnable = (1U << 17),
kPWM_Fault2InterruptEnable = (1U << 18),
kPWM_Fault3InterruptEnable = (1U << 19) }

```

*List of PWM interrupt options.*

- enum `pwm_status_flags_t` {
 

```

kPWM_CompareVal0Flag = (1U << 0),
kPWM_CompareVal1Flag = (1U << 1),
kPWM_CompareVal2Flag = (1U << 2),
kPWM_CompareVal3Flag = (1U << 3),
kPWM_CompareVal4Flag = (1U << 4),
kPWM_CompareVal5Flag = (1U << 5),
kPWM_CaptureX0Flag = (1U << 6),
kPWM_CaptureX1Flag = (1U << 7),
kPWM_CaptureB0Flag = (1U << 8),
kPWM_CaptureB1Flag = (1U << 9),
kPWM_CaptureA0Flag = (1U << 10),
kPWM_CaptureA1Flag = (1U << 11),
kPWM_ReloadFlag = (1U << 12),
kPWM_ReloadErrorFlag = (1U << 13),
kPWM_RegUpdatedFlag = (1U << 14),
kPWM_Fault0Flag = (1U << 16),
kPWM_Fault1Flag = (1U << 17),
kPWM_Fault2Flag = (1U << 18),
kPWM_Fault3Flag = (1U << 19) }

```

*List of PWM status flags.*

- enum `pwm_dma_enable_t` {

```

kPWM_CaptureX0DMAEnable = (1U << 0),
kPWM_CaptureX1DMAEnable = (1U << 1),
kPWM_CaptureB0DMAEnable = (1U << 2),
kPWM_CaptureB1DMAEnable = (1U << 3),
kPWM_CaptureA0DMAEnable = (1U << 4),
kPWM_CaptureA1DMAEnable = (1U << 5) }

```

*List of PWM DMA options.*

- enum `pwm_dma_source_select_t` {  
`kPWM_DMARequestDisable` = 0U,  
`kPWM_DMAWatermarksEnable`,  
`kPWM_DMALocalSync`,  
`kPWM_DMALocalReload` }

*List of PWM capture DMA enable source select.*

- enum `pwm_watermark_control_t` {  
`kPWM_FIFOWatermarksOR` = 0U,  
`kPWM_FIFOWatermarksAND` }

*PWM FIFO Watermark AND Control.*

- enum `pwm_mode_t` {  
`kPWM_SignedCenterAligned` = 0U,  
`kPWM_CenterAligned`,  
`kPWM_SignedEdgeAligned`,  
`kPWM_EdgeAligned` }

*PWM operation mode.*

- enum `pwm_level_select_t` {  
`kPWM_HighTrue` = 0U,  
`kPWM_LowTrue` }

*PWM output pulse mode, high-true or low-true.*

- enum `pwm_fault_state_t` {  
`kPWM_PwmFaultState0`,  
`kPWM_PwmFaultState1`,  
`kPWM_PwmFaultState2`,  
`kPWM_PwmFaultState3` }

*PWM output fault status.*

- enum `pwm_reload_source_select_t` {  
`kPWM_LocalReload` = 0U,  
`kPWM_MasterReload` }

*PWM reload source select.*

- enum `pwm_fault_clear_t` {  
`kPWM_Automatic` = 0U,  
`kPWM_ManualNormal`,  
`kPWM_ManualSafety` }

*PWM fault clearing options.*

- enum `pwm_module_control_t` {  
`kPWM_Control_Module_0` = (1U << 0),  
`kPWM_Control_Module_1` = (1U << 1),  
`kPWM_Control_Module_2` = (1U << 2),  
`kPWM_Control_Module_3` = (1U << 3) }

*Options for submodule master control operation.*

## Functions

- void `PWM_SetupInputCapture` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, const `pwm_input_capture_param_t` \*inputCaptureParams)  
*Sets up the PWM input capture.*
- void `PWM_SetupFaultInputFilter` (PWM\_Type \*base, const `pwm_fault_input_filter_param_t` \*faultInputFilterParams)  
*Sets up the PWM fault input filter.*
- void `PWM_SetupFaults` (PWM\_Type \*base, `pwm_fault_input_t` faultNum, const `pwm_fault_param_t` \*faultParams)  
*Sets up the PWM fault protection.*
- void `PWM_FaultDefaultConfig` (`pwm_fault_param_t` \*config)  
*Fill in the PWM fault config struct with the default settings.*
- void `PWM_SetupForceSignal` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, `pwm_force_signal_t` mode)  
*Selects the signal to output on a PWM pin when a FORCE\_OUT signal is asserted.*
- static void `PWM_OutputTriggerEnable` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_value_register_t` valueRegister, bool activate)  
*Enables or disables the PWM output trigger.*
- static void `PWM_ActivateOutputTrigger` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint16\_t valueRegisterMask)  
*Enables the PWM output trigger.*
- static void `PWM_DeactivateOutputTrigger` (PWM\_Type \*base, `pwm_submodule_t` subModule, uint16\_t valueRegisterMask)  
*Disables the PWM output trigger.*
- static void `PWM_SetupSwCtrlOut` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, bool value)  
*Sets the software control output for a pin to high or low.*
- static void `PWM_SetPwmLdok` (PWM\_Type \*base, uint8\_t subModulesToUpdate, bool value)  
*Sets or clears the PWM LDOK bit on a single or multiple submodules.*
- static void `PWM_SetPwmFaultState` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, `pwm_fault_state_t` faultState)  
*Set PWM output fault status.*
- static void `PWM_SetupFaultDisableMap` (PWM\_Type \*base, `pwm_submodule_t` subModule, `pwm_channels_t` pwmChannel, `pwm_fault_channels_t` pwm\_fault\_channels, uint16\_t value)  
*Set PWM fault disable mapping.*

## Driver version

- #define `FSL_PWM_DRIVER_VERSION` (MAKE\_VERSION(2, 2, 1))  
*Version 2.2.1.*

## Initialization and deinitialization

- `status_t PWM_Init` (PWM\_Type \*base, `pwm_submodule_t` subModule, const `pwm_config_t` \*config)  
*Un gates the PWM submodule clock and configures the peripheral for basic operation.*



- void **PWM\_Deinit** (PWM\_Type \*base, **pwm\_submodule\_t** subModule)  
*Gate the PWM submodule clock.*
- void **PWM\_GetDefaultConfig** (**pwm\_config\_t** \*config)  
*Fill in the PWM config struct with the default settings.*

## Module PWM output

- **status\_t PWM\_SetupPwm** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, const **pwm\_signal\_param\_t** \*chnlParams, uint8\_t numOfChnls, **pwm\_mode\_t** mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz)  
*Sets up the PWM signals for a PWM submodule.*
- void **PWM\_UpdatePwmDutycycle** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, **pwm\_channels\_t** pwmSignal, **pwm\_mode\_t** currPwmMode, uint8\_t dutyCyclePercent)  
*Updates the PWM signal's dutycycle.*
- void **PWM\_UpdatePwmDutycycleHighAccuracy** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, **pwm\_channels\_t** pwmSignal, **pwm\_mode\_t** currPwmMode, uint16\_t dutyCycle)  
*Updates the PWM signal's dutycycle with 16-bit accuracy.*

## Interrupts Interface

- void **PWM\_EnableInterrupts** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, uint32\_t mask)  
*Enables the selected PWM interrupts.*
- void **PWM\_DisableInterrupts** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, uint32\_t mask)  
*Disables the selected PWM interrupts.*
- uint32\_t **PWM\_GetEnabledInterrupts** (PWM\_Type \*base, **pwm\_submodule\_t** subModule)  
*Gets the enabled PWM interrupts.*

## DMA Interface

- static void **PWM\_DMAFIFOWatermarkControl** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, **pwm\_watermark\_control\_t** pwm\_watermark\_control)  
*Capture DMA Enable Source Select.*
- static void **PWM\_DMACaptureSourceSelect** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, **pwm\_dma\_source\_select\_t** pwm\_dma\_source\_select)  
*Capture DMA Enable Source Select.*
- static void **PWM\_EnableDMACapture** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, uint16\_t mask, bool activate)  
*Enables or disables the selected PWM DMA Capture read request.*
- static void **PWM\_EnableDMAWrite** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, bool activate)  
*Enables or disables the PWM DMA write request.*

## Status Interface

- uint32\_t **PWM\_GetStatusFlags** (PWM\_Type \*base, **pwm\_submodule\_t** subModule)  
*Gets the PWM status flags.*
- void **PWM\_ClearStatusFlags** (PWM\_Type \*base, **pwm\_submodule\_t** subModule, uint32\_t mask)  
*Clears the PWM status flags.*

## Timer Start and Stop

- static void [PWM\\_StartTimer](#) (PWM\_Type \*base, uint8\_t subModulesToStart)  
*Starts the PWM counter for a single or multiple submodules.*
- static void [PWM\\_StopTimer](#) (PWM\_Type \*base, uint8\_t subModulesToStop)  
*Stops the PWM counter for a single or multiple submodules.*

## 51.5 Data Structure Documentation

### 51.5.1 struct pwm\_signal\_param\_t

#### Data Fields

- [pwm\\_channels\\_t](#) pwmChannel  
*PWM channel being configured; PWM A or PWM B.*
- uint8\_t [dutyCyclePercent](#)  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*
- [pwm\\_level\\_select\\_t](#) level  
*PWM output active level select.*
- uint16\_t [deadtimeValue](#)  
*The deadtime value; only used if channel pair is operating in complementary mode.*
- [pwm\\_fault\\_state\\_t](#) faultState  
*PWM output fault status.*

#### Field Documentation

##### (1) uint8\_t pwm\_signal\_param\_t::dutyCyclePercent

100=always active signal (100% duty cycle)

### 51.5.2 struct pwm\_config\_t

This structure holds the configuration settings for the PWM peripheral. To initialize this structure to reasonable defaults, call the [PWM\\_GetDefaultConfig\(\)](#) function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

#### Data Fields

- bool [enableDebugMode](#)  
*true: PWM continues to run in debug mode; false: PWM is paused in debug mode*
- bool [enableWait](#)  
*true: PWM continues to run in WAIT mode; false: PWM is paused in WAIT mode*
- [pwm\\_init\\_source\\_t](#) initializationControl  
*Option to initialize the counter.*
- [pwm\\_clock\\_source\\_t](#) clockSource

- *Clock source for the counter.*  
`pwm_clock_prescale_t prescale`
- *Pre-scaler to divide down the clock.*  
`pwm_chnl_pair_operation_t pairOperation`
- *Channel pair in independent or complementary mode.*  
`pwm_register_reload_t reloadLogic`
- *PWM Reload logic setup.*  
`pwm_reload_source_select_t reloadSelect`
- *Reload source select.*  
`pwm_load_frequency_t reloadFrequency`
- *Specifies when to reload, used when user's choice is not immediate reload.*  
`pwm_force_output_trigger_t forceTrigger`
- *Specify which signal will trigger a FORCE\_OUT.*

### 51.5.3 struct pwm\_fault\_input\_filter\_param\_t

#### Data Fields

- `uint8_t faultFilterCount`  
*Fault filter count.*
- `uint8_t faultFilterPeriod`  
*Fault filter period; value of 0 will bypass the filter.*
- `bool faultGlitchStretch`  
*Fault Glitch Stretch Enable: A logic 1 means that input fault signals will be stretched to at least 2 IPBus clock cycles.*

### 51.5.4 struct pwm\_fault\_param\_t

#### Data Fields

- `pwm_fault_clear_t faultClearingMode`  
*Fault clearing mode to use.*
- `bool faultLevel`  
*true: Logic 1 indicates fault; false: Logic 0 indicates fault*
- `bool enableCombinationalPath`  
*true: Combinational Path from fault input is enabled; false: No combination path is available*
- `pwm_fault_recovery_mode_t recoverMode`  
*Specify when to re-enable the PWM output.*

### 51.5.5 struct pwm\_input\_capture\_param\_t

#### Data Fields

- `bool captureInputSel`  
*true: Use the edge counter signal as source false: Use the raw input signal from the pin as source*

- `uint8_t edgeCompareValue`  
*Compare value, used only if edge counter is used as source.*
- `pwm_input_capture_edge_t edge0`  
*Specify which edge causes a capture for input circuitry 0.*
- `pwm_input_capture_edge_t edge1`  
*Specify which edge causes a capture for input circuitry 1.*
- `bool enableOneShotCapture`  
*true: Use one-shot capture mode; false: Use free-running capture mode*
- `uint8_t fifoWatermark`  
*Watermark level for capture FIFO.*

### Field Documentation

#### (1) `uint8_t pwm_input_capture_param_t::fifoWatermark`

The capture flags in the status register will set if the word count in the FIFO is greater than this watermark level

## 51.6 Enumeration Type Documentation

### 51.6.1 `enum pwm_submodule_t`

Enumerator

- `kPWM_Module_0` Submodule 0.
- `kPWM_Module_1` Submodule 1.
- `kPWM_Module_2` Submodule 2.
- `kPWM_Module_3` Submodule 3.

### 51.6.2 `enum pwm_value_register_t`

Enumerator

- `kPWM_ValueRegister_0` PWM Value0 register.
- `kPWM_ValueRegister_1` PWM Value1 register.
- `kPWM_ValueRegister_2` PWM Value2 register.
- `kPWM_ValueRegister_3` PWM Value3 register.
- `kPWM_ValueRegister_4` PWM Value4 register.
- `kPWM_ValueRegister_5` PWM Value5 register.

### 51.6.3 `enum _pwm_value_register_mask`

Enumerator

- `kPWM_ValueRegisterMask_0` PWM Value0 register mask.

***kPWM\_ValueRegisterMask\_1*** PWM Value1 register mask.  
***kPWM\_ValueRegisterMask\_2*** PWM Value2 register mask.  
***kPWM\_ValueRegisterMask\_3*** PWM Value3 register mask.  
***kPWM\_ValueRegisterMask\_4*** PWM Value4 register mask.  
***kPWM\_ValueRegisterMask\_5*** PWM Value5 register mask.

#### 51.6.4 enum pwm\_clock\_source\_t

Enumerator

***kPWM\_BusClock*** The IPBus clock is used as the clock.  
***kPWM\_ExternalClock*** EXT\_CLK is used as the clock.  
***kPWM\_Submodule0Clock*** Clock of the submodule 0 (AUX\_CLK) is used as the source clock.

#### 51.6.5 enum pwm\_clock\_prescale\_t

Enumerator

***kPWM\_Prescale\_Divide\_1*** PWM clock frequency = fclk/1.  
***kPWM\_Prescale\_Divide\_2*** PWM clock frequency = fclk/2.  
***kPWM\_Prescale\_Divide\_4*** PWM clock frequency = fclk/4.  
***kPWM\_Prescale\_Divide\_8*** PWM clock frequency = fclk/8.  
***kPWM\_Prescale\_Divide\_16*** PWM clock frequency = fclk/16.  
***kPWM\_Prescale\_Divide\_32*** PWM clock frequency = fclk/32.  
***kPWM\_Prescale\_Divide\_64*** PWM clock frequency = fclk/64.  
***kPWM\_Prescale\_Divide\_128*** PWM clock frequency = fclk/128.

#### 51.6.6 enum pwm\_force\_output\_trigger\_t

Enumerator

***kPWM\_Force\_Local*** The local force signal, CTRL2[FORCE], from the submodule is used to force updates.  
***kPWM\_Force\_Master*** The master force signal from submodule 0 is used to force updates.  
***kPWM\_Force\_LocalReload*** The local reload signal from this submodule is used to force updates without regard to the state of LDOK.  
***kPWM\_Force\_MasterReload*** The master reload signal from submodule 0 is used to force updates if LDOK is set.  
***kPWM\_Force\_LocalSync*** The local sync signal from this submodule is used to force updates.  
***kPWM\_Force\_MasterSync*** The master sync signal from submodule0 is used to force updates.  
***kPWM\_Force\_External*** The external force signal, EXT\_FORCE, from outside the PWM module causes updates.

***kPWM\_Force\_ExternalSync*** The external sync signal, EXT\_SYNC, from outside the PWM module causes updates.

### 51.6.7 enum pwm\_init\_source\_t

Enumerator

***kPWM\_Initialize\_LocalSync*** Local sync causes initialization.  
***kPWM\_Initialize\_MasterReload*** Master reload from submodule 0 causes initialization.  
***kPWM\_Initialize\_MasterSync*** Master sync from submodule 0 causes initialization.  
***kPWM\_Initialize\_ExtSync*** EXT\_SYNC causes initialization.

### 51.6.8 enum pwm\_load\_frequency\_t

Enumerator

***kPWM\_LoadEveryOportunity*** Every PWM opportunity.  
***kPWM\_LoadEvery2Oportunity*** Every 2 PWM opportunities.  
***kPWM\_LoadEvery3Oportunity*** Every 3 PWM opportunities.  
***kPWM\_LoadEvery4Oportunity*** Every 4 PWM opportunities.  
***kPWM\_LoadEvery5Oportunity*** Every 5 PWM opportunities.  
***kPWM\_LoadEvery6Oportunity*** Every 6 PWM opportunities.  
***kPWM\_LoadEvery7Oportunity*** Every 7 PWM opportunities.  
***kPWM\_LoadEvery8Oportunity*** Every 8 PWM opportunities.  
***kPWM\_LoadEvery9Oportunity*** Every 9 PWM opportunities.  
***kPWM\_LoadEvery10Oportunity*** Every 10 PWM opportunities.  
***kPWM\_LoadEvery11Oportunity*** Every 11 PWM opportunities.  
***kPWM\_LoadEvery12Oportunity*** Every 12 PWM opportunities.  
***kPWM\_LoadEvery13Oportunity*** Every 13 PWM opportunities.  
***kPWM\_LoadEvery14Oportunity*** Every 14 PWM opportunities.  
***kPWM\_LoadEvery15Oportunity*** Every 15 PWM opportunities.  
***kPWM\_LoadEvery16Oportunity*** Every 16 PWM opportunities.

### 51.6.9 enum pwm\_fault\_input\_t

Enumerator

***kPWM\_Fault\_0*** Fault 0 input pin.  
***kPWM\_Fault\_1*** Fault 1 input pin.  
***kPWM\_Fault\_2*** Fault 2 input pin.  
***kPWM\_Fault\_3*** Fault 3 input pin.

### 51.6.10 enum pwm\_fault\_disable\_t

Enumerator

*kPWM\_FaultDisable\_0* Fault 0 disable mapping.  
*kPWM\_FaultDisable\_1* Fault 1 disable mapping.  
*kPWM\_FaultDisable\_2* Fault 2 disable mapping.  
*kPWM\_FaultDisable\_3* Fault 3 disable mapping.

### 51.6.11 enum pwm\_input\_capture\_edge\_t

Enumerator

*kPWM\_Disable* Disabled.  
*kPWM\_FallingEdge* Capture on falling edge only.  
*kPWM\_RisingEdge* Capture on rising edge only.  
*kPWM\_RiseAndFallEdge* Capture on rising or falling edge.

### 51.6.12 enum pwm\_force\_signal\_t

Enumerator

*kPWM\_UsePwm* Generated PWM signal is used by the deadtime logic.  
*kPWM\_InvertedPwm* Inverted PWM signal is used by the deadtime logic.  
*kPWM\_SoftwareControl* Software controlled value is used by the deadtime logic.  
*kPWM\_UseExternal* PWM\_EXT\_A signal is used by the deadtime logic.

### 51.6.13 enum pwm\_chnl\_pair\_operation\_t

Enumerator

*kPWM\_Independent* PWM A & PWM B operate as 2 independent channels.  
*kPWM\_ComplementaryPwmA* PWM A & PWM B are complementary channels, PWM A generates the signal.  
*kPWM\_ComplementaryPwmB* PWM A & PWM B are complementary channels, PWM B generates the signal.

### 51.6.14 enum pwm\_register\_reload\_t

Enumerator

- kPWM\_ReloadImmediate* Buffered-registers get loaded with new values as soon as LDOK bit is set.
- kPWM\_ReloadPwmHalfCycle* Registers loaded on a PWM half cycle.
- kPWM\_ReloadPwmFullCycle* Registers loaded on a PWM full cycle.
- kPWM\_ReloadPwmHalfAndFullCycle* Registers loaded on a PWM half & full cycle.

### 51.6.15 enum pwm\_fault\_recovery\_mode\_t

Enumerator

- kPWM\_NoRecovery* PWM output will stay inactive.
- kPWM\_RecoverHalfCycle* PWM output re-enabled at the first half cycle.
- kPWM\_RecoverFullCycle* PWM output re-enabled at the first full cycle.
- kPWM\_RecoverHalfAndFullCycle* PWM output re-enabled at the first half or full cycle.

### 51.6.16 enum pwm\_interrupt\_enable\_t

Enumerator

- kPWM\_CompareVal0InterruptEnable* PWM VAL0 compare interrupt.
- kPWM\_CompareVal1InterruptEnable* PWM VAL1 compare interrupt.
- kPWM\_CompareVal2InterruptEnable* PWM VAL2 compare interrupt.
- kPWM\_CompareVal3InterruptEnable* PWM VAL3 compare interrupt.
- kPWM\_CompareVal4InterruptEnable* PWM VAL4 compare interrupt.
- kPWM\_CompareVal5InterruptEnable* PWM VAL5 compare interrupt.
- kPWM\_CaptureX0InterruptEnable* PWM capture X0 interrupt.
- kPWM\_CaptureX1InterruptEnable* PWM capture X1 interrupt.
- kPWM\_CaptureB0InterruptEnable* PWM capture B0 interrupt.
- kPWM\_CaptureB1InterruptEnable* PWM capture B1 interrupt.
- kPWM\_CaptureA0InterruptEnable* PWM capture A0 interrupt.
- kPWM\_CaptureA1InterruptEnable* PWM capture A1 interrupt.
- kPWM\_ReloadInterruptEnable* PWM reload interrupt.
- kPWM\_ReloadErrorInterruptEnable* PWM reload error interrupt.
- kPWM\_Fault0InterruptEnable* PWM fault 0 interrupt.
- kPWM\_Fault1InterruptEnable* PWM fault 1 interrupt.
- kPWM\_Fault2InterruptEnable* PWM fault 2 interrupt.
- kPWM\_Fault3InterruptEnable* PWM fault 3 interrupt.



### 51.6.17 enum pwm\_status\_flags\_t

Enumerator

***kPWM\_CompareVal0Flag*** PWM VAL0 compare flag.  
***kPWM\_CompareVal1Flag*** PWM VAL1 compare flag.  
***kPWM\_CompareVal2Flag*** PWM VAL2 compare flag.  
***kPWM\_CompareVal3Flag*** PWM VAL3 compare flag.  
***kPWM\_CompareVal4Flag*** PWM VAL4 compare flag.  
***kPWM\_CompareVal5Flag*** PWM VAL5 compare flag.  
***kPWM\_CaptureX0Flag*** PWM capture X0 flag.  
***kPWM\_CaptureX1Flag*** PWM capture X1 flag.  
***kPWM\_CaptureB0Flag*** PWM capture B0 flag.  
***kPWM\_CaptureB1Flag*** PWM capture B1 flag.  
***kPWM\_CaptureA0Flag*** PWM capture A0 flag.  
***kPWM\_CaptureA1Flag*** PWM capture A1 flag.  
***kPWM\_ReloadFlag*** PWM reload flag.  
***kPWM\_ReloadErrorFlag*** PWM reload error flag.  
***kPWM\_RegUpdatedFlag*** PWM registers updated flag.  
***kPWM\_Fault0Flag*** PWM fault 0 flag.  
***kPWM\_Fault1Flag*** PWM fault 1 flag.  
***kPWM\_Fault2Flag*** PWM fault 2 flag.  
***kPWM\_Fault3Flag*** PWM fault 3 flag.

### 51.6.18 enum pwm\_dma\_enable\_t

Enumerator

***kPWM\_CaptureX0DMAEnable*** PWM capture X0 DMA.  
***kPWM\_CaptureX1DMAEnable*** PWM capture X1 DMA.  
***kPWM\_CaptureB0DMAEnable*** PWM capture B0 DMA.  
***kPWM\_CaptureB1DMAEnable*** PWM capture B1 DMA.  
***kPWM\_CaptureA0DMAEnable*** PWM capture A0 DMA.  
***kPWM\_CaptureA1DMAEnable*** PWM capture A1 DMA.

### 51.6.19 enum pwm\_dma\_source\_select\_t

Enumerator

***kPWM\_DMAResourceDisable*** Read DMA requests disabled.  
***kPWM\_DMAWatermarksEnable*** Exceeding a FIFO watermark sets the DMA read request.  
***kPWM\_DMALocalSync*** A local sync (VAL1 matches counter) sets the read DMA request.  
***kPWM\_DMALocalReload*** A local reload (STS[RF] being set) sets the read DMA request.

**51.6.20 enum pwm\_watermark\_control\_t**

Enumerator

***kPWM\_FIFOWatermarksOR*** Selected FIFO watermarks are OR'ed together.

***kPWM\_FIFOWatermarksAND*** Selected FIFO watermarks are AND'ed together.

**51.6.21 enum pwm\_mode\_t**

Enumerator

***kPWM\_SignedCenterAligned*** Signed center-aligned.

***kPWM\_CenterAligned*** Unsigned center-aligned.

***kPWM\_SignedEdgeAligned*** Signed edge-aligned.

***kPWM\_EdgeAligned*** Unsigned edge-aligned.

**51.6.22 enum pwm\_level\_select\_t**

Enumerator

***kPWM\_HighTrue*** High level represents "on" or "active" state.

***kPWM\_LowTrue*** Low level represents "on" or "active" state.

**51.6.23 enum pwm\_fault\_state\_t**

Enumerator

***kPWM\_PwmFaultState0*** Output is forced to logic 0 state prior to consideration of output polarity control.

***kPWM\_PwmFaultState1*** Output is forced to logic 1 state prior to consideration of output polarity control.

***kPWM\_PwmFaultState2*** Output is tristated.

***kPWM\_PwmFaultState3*** Output is tristated.

**51.6.24 enum pwm\_reload\_source\_select\_t**

Enumerator

***kPWM\_LocalReload*** The local reload signal is used to reload registers.

***kPWM\_MasterReload*** The master reload signal (from submodule 0) is used to reload.

### 51.6.25 enum pwm\_fault\_clear\_t

Enumerator

*kPWM\_Automatic* Automatic fault clearing.  
*kPWM\_ManualNormal* Manual fault clearing with no fault safety mode.  
*kPWM\_ManualSafety* Manual fault clearing with fault safety mode.

### 51.6.26 enum pwm\_module\_control\_t

Enumerator

*kPWM\_Control\_Module\_0* Control submodule 0's start/stop,buffer reload operation.  
*kPWM\_Control\_Module\_1* Control submodule 1's start/stop,buffer reload operation.  
*kPWM\_Control\_Module\_2* Control submodule 2's start/stop,buffer reload operation.  
*kPWM\_Control\_Module\_3* Control submodule 3's start/stop,buffer reload operation.

## 51.7 Function Documentation

### 51.7.1 status\_t PWM\_Init ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, const pwm\_config\_t \* *config* )

Note

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

Parameters

|                  |                                         |
|------------------|-----------------------------------------|
| <i>base</i>      | PWM peripheral base address             |
| <i>subModule</i> | PWM submodule to configure              |
| <i>config</i>    | Pointer to user's PWM config structure. |

Returns

kStatus\_Success means success; else failed.

### 51.7.2 void PWM\_Deinit ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )

## Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>base</i>      | PWM peripheral base address   |
| <i>subModule</i> | PWM submodule to deinitialize |

### 51.7.3 void PWM\_GetDefaultConfig ( pwm\_config\_t \* config )

The default values are:

```
* config->enableDebugMode = false;
* config->enableWait = false;
* config->reloadSelect = kPWM_LocalReload;
* config->clockSource = kPWM_BusClock;
* config->prescale = kPWM_Prescale_Divide_1;
* config->initializationControl = kPWM_Initialize_LocalSync;
* config->forceTrigger = kPWM_Force_Local;
* config->reloadFrequency = kPWM_LoadEveryOpportunity;
* config->reloadLogic = kPWM_ReloadImmediate;
* config->pairOperation = kPWM_Independent;
*
```

## Parameters

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

### 51.7.4 status\_t PWM\_SetupPwm ( PWM\_Type \* base, pwm\_submodule\_t subModule, const pwm\_signal\_param\_t \* chnlParams, uint8\_t numOfChnls, pwm\_mode\_t mode, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz )

The function initializes the submodule according to the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. If the dead time insertion logic is enabled, the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

|                    |                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <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. Array size should not be more than 2 as each submodule has 2 pins to output PWM |
| <i>mode</i>        | PWM operation mode, options available in enumeration <a href="#">pwm_mode_t</a>                                                                                  |
| <i>pwmFreq_Hz</i>  | PWM signal frequency in Hz                                                                                                                                       |
| <i>srcClock_Hz</i> | PWM main counter clock in Hz.                                                                                                                                    |

## Returns

Returns kStatusFail if there was error setting up the signal; kStatusSuccess otherwise

### 51.7.5 void PWM\_UpdatePwmDutycycle ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmSignal*, pwm\_mode\_t *currPwmMode*, uint8\_t *dutyCyclePercent* )

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                                   |
| <i>subModule</i>         | PWM submodule to configure                                                                                                    |
| <i>pwmSignal</i>         | Signal (PWM A or PWM B) to update                                                                                             |
| <i>currPwmMode</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) |

### 51.7.6 void PWM\_UpdatePwmDutycycleHighAccuracy ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmSignal*, pwm\_mode\_t *currPwmMode*, uint16\_t *dutyCycle* )

The function updates the PWM dutycyle to the new value that is passed in. If the dead time insertion logic is enabled then the pulse period is reduced by the dead time period specified by the user.

## Parameters

|                    |                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | PWM peripheral base address                                                                                                       |
| <i>subModule</i>   | PWM submodule to configure                                                                                                        |
| <i>pwmSignal</i>   | Signal (PWM A or PWM B) to update                                                                                                 |
| <i>currPwmMode</i> | The current PWM mode set during PWM setup                                                                                         |
| <i>dutyCycle</i>   | New PWM pulse width, value should be between 0 to 65535 0=inactive signal(0% duty cycle)... 65535=active signal (100% duty cycle) |

**51.7.7 void PWM\_SetupInputCapture ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, const pwm\_input\_capture\_param\_t \* *inputCaptureParams* )**

Each PWM submodule has 3 pins that can be configured for use as input capture pins. This function sets up the capture parameters for each pin and enables the pin for input capture operation.

## Parameters

|                            |                                              |
|----------------------------|----------------------------------------------|
| <i>base</i>                | PWM peripheral base address                  |
| <i>subModule</i>           | PWM submodule to configure                   |
| <i>pwmChannel</i>          | Channel in the submodule to setup            |
| <i>inputCapture-Params</i> | Parameters passed in to set up the input pin |

**51.7.8 void PWM\_SetupFaultInputFilter ( PWM\_Type \* *base*, const pwm\_fault\_input\_filter\_param\_t \* *faultInputFilterParams* )**

## Parameters

|                                |                                                        |
|--------------------------------|--------------------------------------------------------|
| <i>base</i>                    | PWM peripheral base address                            |
| <i>faultInput-FilterParams</i> | Parameters passed in to set up the fault input filter. |

**51.7.9 void PWM\_SetupFaults ( PWM\_Type \* *base*, pwm\_fault\_input\_t *faultNum*, const pwm\_fault\_param\_t \* *faultParams* )**

PWM has 4 fault inputs.

## Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | PWM peripheral base address               |
| <i>faultNum</i>    | PWM fault to configure.                   |
| <i>faultParams</i> | Pointer to the PWM fault config structure |

### 51.7.10 void PWM\_FaultDefaultConfig ( pwm\_fault\_param\_t \* *config* )

The default values are:

```
* config->faultClearingMode = kPWM_Automatic;
* config->faultLevel = false;
* config->enableCombinationalPath = true;
* config->recoverMode = kPWM_NoRecovery;
*
```

## Parameters

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

### 51.7.11 void PWM\_SetupForceSignal ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, pwm\_force\_signal\_t *mode* )

The user specifies which channel to configure by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

## Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                    |
| <i>subModule</i>  | PWM submodule to configure                     |
| <i>pwmChannel</i> | Channel to configure                           |
| <i>mode</i>       | Signal to output when a FORCE_OUT is triggered |

### 51.7.12 void PWM\_EnableInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )

## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                         |
| <i>subModule</i> | PWM submodule to configure                                                                                          |
| <i>mask</i>      | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a> |

**51.7.13 void PWM\_DisableInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )**

## Parameters

|                  |                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                         |
| <i>subModule</i> | PWM submodule to configure                                                                                          |
| <i>mask</i>      | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">pwm_interrupt_enable_t</a> |

**51.7.14 uint32\_t PWM\_GetEnabledInterrupts ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )**

## Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

## Returns

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

**51.7.15 static void PWM\_DMAFIFOWatermarkControl ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_watermark\_control\_t *pwm\_watermark\_control* ) [inline], [static]**



## Parameters

|                                          |                                |
|------------------------------------------|--------------------------------|
| <i>base</i>                              | PWM peripheral base address    |
| <i>subModule</i>                         | PWM submodule to configure     |
| <i>pwm_-<br/>watermark_-<br/>control</i> | PWM FIFO watermark and control |

**51.7.16 static void PWM\_DMACaptureSourceSelect ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_dma\_source\_select\_t *pwm\_dma\_source\_select* ) [inline], [static]**

## Parameters

|                                    |                                      |
|------------------------------------|--------------------------------------|
| <i>base</i>                        | PWM peripheral base address          |
| <i>subModule</i>                   | PWM submodule to configure           |
| <i>pwm_dma_-<br/>source_select</i> | PWM capture DMA enable source select |

**51.7.17 static void PWM\_EnableDMACapture ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint16\_t *mask*, bool *activate* ) [inline], [static]**

## Parameters

|                  |                                                                                                                   |
|------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                       |
| <i>subModule</i> | PWM submodule to configure                                                                                        |
| <i>mask</i>      | The DMA to enable or disable. This is a logical OR of members of the enumeration <a href="#">pwm_dma_enable_t</a> |
| <i>activate</i>  | true: Enable DMA read request; false: Disable DMA read request                                                    |

**51.7.18 static void PWM\_EnableDMAWrite ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, bool *activate* ) [inline], [static]**

## Parameters

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                      |
| <i>subModule</i> | PWM submodule to configure                                       |
| <i>activate</i>  | true: Enable DMA write request; false: Disable DMA write request |

### 51.7.19 uint32\_t PWM\_GetStatusFlags ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule* )

## Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>base</i>      | PWM peripheral base address |
| <i>subModule</i> | PWM submodule to configure  |

## Returns

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

### 51.7.20 void PWM\_ClearStatusFlags ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint32\_t *mask* )

## Parameters

|                  |                                                                                                                  |
|------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | PWM peripheral base address                                                                                      |
| <i>subModule</i> | PWM submodule to configure                                                                                       |
| <i>mask</i>      | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">pwm_status_flags_t</a> |

### 51.7.21 static void PWM\_StartTimer ( PWM\_Type \* *base*, uint8\_t *subModulesToStart* ) [inline], [static]

Sets the Run bit which enables the clocks to the PWM submodule. This function can start multiple submodules at the same time.

## Parameters

|                           |                                                                                                                   |
|---------------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                       |
| <i>subModulesTo-Start</i> | PWM submodules to start. This is a logical OR of members of the enumeration <a href="#">pwm-_module_control_t</a> |

### 51.7.22 static void PWM\_StopTimer ( PWM\_Type \* *base*, uint8\_t *subModulesToStop* ) [inline], [static]

Clears the Run bit which resets the submodule's counter. This function can stop multiple submodules at the same time.

## Parameters

|                          |                                                                                                                  |
|--------------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | PWM peripheral base address                                                                                      |
| <i>subModulesTo-Stop</i> | PWM submodules to stop. This is a logical OR of members of the enumeration <a href="#">pwm-_module_control_t</a> |

### 51.7.23 static void PWM\_OutputTriggerEnable ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_value\_register\_t *valueRegister*, bool *activate* ) [inline], [static]

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. Trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. Trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5 register.

## Parameters

|                      |                                                      |
|----------------------|------------------------------------------------------|
| <i>base</i>          | PWM peripheral base address                          |
| <i>subModule</i>     | PWM submodule to configure                           |
| <i>valueRegister</i> | Value register that will activate the trigger        |
| <i>activate</i>      | true: Enable the trigger; false: Disable the trigger |

### 51.7.24 static void PWM\_ActivateOutputTrigger ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint16\_t *valueRegisterMask* ) [inline], [static]

This function allows the user to enable one or more (VAL0-5) PWM trigger.

## Parameters

|                           |                                                                                                                          |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                              |
| <i>subModule</i>          | PWM submodule to configure                                                                                               |
| <i>valueRegister-Mask</i> | Value register mask that will activate one or more (VAL0-5) trigger enumeration <a href="#">_pwm_value_register_mask</a> |

**51.7.25 static void PWM\_DeactivateOutputTrigger ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, uint16\_t *valueRegisterMask* ) [inline], [static]**

This function allows the user to disables one or more (VAL0-5) PWM trigger.

## Parameters

|                           |                                                                                                                            |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                                |
| <i>subModule</i>          | PWM submodule to configure                                                                                                 |
| <i>valueRegister-Mask</i> | Value register mask that will Deactivate one or more (VAL0-5) trigger enumeration <a href="#">_pwm_value_register_mask</a> |

**51.7.26 static void PWM\_SetupSwCtrlOut ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, bool *value* ) [inline], [static]**

The user specifies which channel to modify by supplying the submodule number and whether to modify PWM A or PWM B within that submodule.

## Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>base</i>       | PWM peripheral base address                      |
| <i>subModule</i>  | PWM submodule to configure                       |
| <i>pwmChannel</i> | Channel to configure                             |
| <i>value</i>      | true: Supply a logic 1, false: Supply a logic 0. |

### 51.7.27 static void PWM\_SetPwmLdok ( PWM\_Type \* *base*, uint8\_t *subModulesToUpdate*, bool *value* ) [inline], [static]

Set LDOK bit to load buffered values into CTRL[PRSC] and the INIT, FRACVAL and VAL registers. The values are loaded immediately if kPWM\_ReloadImmediate option was choosen during config. Else the values are loaded at the next PWM reload point. This function can issue the load command to multiple submodules at the same time.

Parameters

|                           |                                                                                                                                        |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                                                                            |
| <i>subModulesToUpdate</i> | PWM submodules to update with buffered values. This is a logical OR of members of the enumeration <a href="#">pwm_module_control_t</a> |
| <i>value</i>              | true: Set LDOK bit for the submodule list; false: Clear LDOK bit                                                                       |

### 51.7.28 static void PWM\_SetPwmFaultState ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, pwm\_fault\_state\_t *faultState* ) [inline], [static]

These bits determine the fault state for the PWM\_A output in fault conditions and STOP mode. It may also define the output state in WAIT and DEBUG modes depending on the settings of CTRL2[WAITEN] and CTRL2[DBGEN]. This function can update PWM output fault status.

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | PWM peripheral base address |
| <i>subModule</i>  | PWM submodule to configure  |
| <i>pwmChannel</i> | Channel to configure        |
| <i>faultState</i> | PWM output fault status     |

### 51.7.29 static void PWM\_SetupFaultDisableMap ( PWM\_Type \* *base*, pwm\_submodule\_t *subModule*, pwm\_channels\_t *pwmChannel*, pwm\_fault\_channels\_t *pwm\_fault\_channels*, uint16\_t *value* ) [inline], [static]

Each of the four bits of this read/write field is one-to-one associated with the four FAULTx inputs of fault channel 0/1. The PWM output will be turned off if there is a logic 1 on an FAULTx input and a 1 in the corresponding bit of this field. A reset sets all bits in this field.

## Parameters

|                           |                                                                                  |
|---------------------------|----------------------------------------------------------------------------------|
| <i>base</i>               | PWM peripheral base address                                                      |
| <i>subModule</i>          | PWM submodule to configure                                                       |
| <i>pwmChannel</i>         | PWM channel to configure                                                         |
| <i>pwm_fault_channels</i> | PWM fault channel to configure                                                   |
| <i>value</i>              | Fault disable mapping mask value enumeration <a href="#">pwm_fault_disable_t</a> |

# Chapter 52

## PXP: Pixel Pipeline

### 52.1 Overview

The MCUXpresso SDK provides a driver for the Pixel Pipeline (PXP)

The PXP is used to process graphics buffers or composite video and graphics data before sending to an LCD display or TV encoder. The PXP driver only provides functional APIs. It does not maintain software level state, so that the APIs could be involved directly to any upper layer graphics framework easily.

To use the PXP driver, call [PXP\\_Init](#) first to enable and initialize the peripheral. Generally, call the PXP driver APIs to configure input buffer, output buffer, and other settings such as flip, rotate, then call [PXP\\_Start](#), thus the PXP starts the processing. When finished, the flag [kPXP\\_CompleteFlag](#) asserts. PXP also supports operation queuing, it means that a new operation could be submitted to PXP while the current PXP operation is running. When current operation finished, the new operation configurations are loaded to PXP register and new processing starts.

### 52.2 Typical use case

#### 52.2.1 PXP normal operation

This example shows how to perform vertical flip to process surface and save to output buffer. The input and output buffer pixel format are RGB888.

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

#### 52.2.2 PXP operation queue

This example shows how to perform vertical flip to process surface using operation queue. The input and output buffer pixel format are RGB888.

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

### Data Structures

- struct [pxp\\_output\\_buffer\\_config\\_t](#)  
*PXP output buffer configuration. [More...](#)*
- struct [pxp\\_ps\\_buffer\\_config\\_t](#)  
*PXP process surface buffer configuration. [More...](#)*
- struct [pxp\\_as\\_buffer\\_config\\_t](#)  
*PXP alpha surface buffer configuration. [More...](#)*
- struct [pxp\\_as\\_blend\\_config\\_t](#)  
*PXP alpha surface blending configuration. [More...](#)*
- struct [pxp\\_csc2\\_config\\_t](#)

- *PXP CSC2 configuration. [More...](#)*
- struct [pxp\\_dither\\_final\\_lut\\_data\\_t](#)  
*PXP dither final LUT data. [More...](#)*
- struct [pxp\\_dither\\_config\\_t](#)  
*PXP dither configuration. [More...](#)*
- struct [pxp\\_porter\\_duff\\_config\\_t](#)  
*PXP Porter Duff configuration. [More...](#)*
- struct [pxp\\_pic\\_copy\\_config\\_t](#)  
*PXP Porter Duff blend mode. [More...](#)*

## Enumerations

- enum [\\_pxp\\_interrupt\\_enable](#) {  
  [kPXP\\_CommandLoadInterruptEnable](#) = PXP\_CTRL\_NEXT\_IRQ\_ENABLE\_MASK,  
  [kPXP\\_CompleteInterruptEnable](#) = PXP\_CTRL\_IRQ\_ENABLE\_MASK }  
*PXP interrupts to enable.*
- enum [\\_pxp\\_flags](#) {  
  [kPXP\\_CommandLoadFlag](#) = PXP\_STAT\_NEXT\_IRQ\_MASK,  
  [kPXP\\_CompleteFlag](#) = PXP\_STAT\_IRQ0\_MASK,  
  [kPXP\\_Axi0ReadErrorFlag](#) = PXP\_STAT\_AXI\_READ\_ERROR\_0\_MASK,  
  [kPXP\\_Axi0WriteErrorFlag](#) = PXP\_STAT\_AXI\_WRITE\_ERROR\_0\_MASK }  
*PXP status flags.*
- enum [pxp\\_flip\\_mode\\_t](#) {  
  [kPXP\\_FlipDisable](#) = 0U,  
  [kPXP\\_FlipHorizontal](#) = 0x01U,  
  [kPXP\\_FlipVertical](#) = 0x02U,  
  [kPXP\\_FlipBoth](#) = 0x03U }  
*PXP output flip mode.*
- enum [pxp\\_rotate\\_position\\_t](#) {  
  [kPXP\\_RotateOutputBuffer](#) = 0U,  
  [kPXP\\_RotateProcessSurface](#) }  
*PXP rotate mode.*
- enum [pxp\\_rotate\\_degree\\_t](#) {  
  [kPXP\\_Rotate0](#) = 0U,  
  [kPXP\\_Rotate90](#),  
  [kPXP\\_Rotate180](#),  
  [kPXP\\_Rotate270](#) }  
*PXP rotate degree.*
- enum [pxp\\_interlaced\\_output\\_mode\\_t](#) {  
  [kPXP\\_OutputProgressive](#) = 0U,  
  [kPXP\\_OutputField0](#),  
  [kPXP\\_OutputField1](#),  
  [kPXP\\_OutputInterlaced](#) }  
*PXP interlaced output mode.*
- enum [pxp\\_output\\_pixel\\_format\\_t](#) {



```

kPXP_OutputPixelFormatARGB8888 = 0x0,
kPXP_OutputPixelFormatRGB888 = 0x4,
kPXP_OutputPixelFormatRGB888P = 0x5,
kPXP_OutputPixelFormatARGB1555 = 0x8,
kPXP_OutputPixelFormatARGB4444 = 0x9,
kPXP_OutputPixelFormatRGB555 = 0xC,
kPXP_OutputPixelFormatRGB444 = 0xD,
kPXP_OutputPixelFormatRGB565 = 0xE,
kPXP_OutputPixelFormatYUV1P444 = 0x10,
kPXP_OutputPixelFormatUYVY1P422 = 0x12,
kPXP_OutputPixelFormatVYUY1P422 = 0x13,
kPXP_OutputPixelFormatY8 = 0x14,
kPXP_OutputPixelFormatY4 = 0x15,
kPXP_OutputPixelFormatYUV2P422 = 0x18,
kPXP_OutputPixelFormatYUV2P420 = 0x19,
kPXP_OutputPixelFormatYVU2P422 = 0x1A,
kPXP_OutputPixelFormatYVU2P420 = 0x1B }

```

*PXP output buffer format.*

- enum `pxp_ps_pixel_format_t` {
 

```

kPXP_PsPixelFormatRGB888 = 0x4,
kPXP_PsPixelFormatRGB555 = 0xC,
kPXP_PsPixelFormatRGB444 = 0xD,
kPXP_PsPixelFormatRGB565 = 0xE,
kPXP_PsPixelFormatYUV1P444 = 0x10,
kPXP_PsPixelFormatUYVY1P422 = 0x12,
kPXP_PsPixelFormatVYUY1P422 = 0x13,
kPXP_PsPixelFormatY8 = 0x14,
kPXP_PsPixelFormatY4 = 0x15,
kPXP_PsPixelFormatYUV2P422 = 0x18,
kPXP_PsPixelFormatYUV2P420 = 0x19,
kPXP_PsPixelFormatYVU2P422 = 0x1A,
kPXP_PsPixelFormatYVU2P420 = 0x1B,
kPXP_PsPixelFormatYVU422 = 0x1E,
kPXP_PsPixelFormatYVU420 = 0x1F }

```

*PXP process surface buffer pixel format.*

- enum `pxp_ps_yuv_format_t` {
 

```

kPXP_PsYUVFormatYUV = 0U,
kPXP_PsYUVFormatYCbCr }

```

*PXP process surface buffer YUV format.*

- enum `pxp_as_pixel_format_t` {

```

kPXP_AsPixelFormatARGB8888 = 0x0,
kPXP_AsPixelFormatRGB888 = 0x4,
kPXP_AsPixelFormatARGB1555 = 0x8,
kPXP_AsPixelFormatARGB4444 = 0x9,
kPXP_AsPixelFormatRGB555 = 0xC,
kPXP_AsPixelFormatRGB444 = 0xD,
kPXP_AsPixelFormatRGB565 = 0xE }

```

*PXP alpha surface buffer pixel format.*

- enum `pxp_alpha_mode_t` {  
`kPXP_AlphaEmbedded`,  
`kPXP_AlphaOverride`,  
`kPXP_AlphaMultiply`,  
`kPXP_AlphaRop` }

*PXP alpha mode during blending.*

- enum `pxp_rop_mode_t` {  
`kPXP_RopMaskAs` = 0x0,  
`kPXP_RopMaskNotAs` = 0x1,  
`kPXP_RopMaskAsNot` = 0x2,  
`kPXP_RopMergeAs` = 0x3,  
`kPXP_RopMergeNotAs` = 0x4,  
`kPXP_RopMergeAsNot` = 0x5,  
`kPXP_RopNotCopyAs` = 0x6,  
`kPXP_RopNot` = 0x7,  
`kPXP_RopNotMaskAs` = 0x8,  
`kPXP_RopNotMergeAs` = 0x9,  
`kPXP_RopXorAs` = 0xA,  
`kPXP_RopNotXorAs` = 0xB }

*PXP ROP mode during blending.*

- enum `pxp_block_size_t` {  
`kPXP_BlockSize8` = 0U,  
`kPXP_BlockSize16` }

*PXP process block size.*

- enum `pxp_csc1_mode_t` {  
`kPXP_Csc1YUV2RGB` = 0U,  
`kPXP_Csc1YCbCr2RGB` }

*PXP CSC1 mode.*

- enum `pxp_csc2_mode_t` {  
`kPXP_Csc2YUV2RGB` = 0U,  
`kPXP_Csc2YCbCr2RGB`,  
`kPXP_Csc2RGB2YUV`,  
`kPXP_Csc2RGB2YCbCr` }

*PXP CSC2 mode.*

- enum `pxp_ram_t` {  
`kPXP_RamDither0Lut` = 0U,  
`kPXP_RamDither1Lut` = 3U,  
`kPXP_RamDither2Lut` = 4U }

- PXP internal memory.*
  - enum `_pxp_dither_mode` {  
`kPXP_DitherPassThrough` = 0U,  
`kPXP_DitherOrdered` = 3U,  
`kPXP_DitherQuantOnly` = 4U }
- PXP dither mode.*
  - enum `_pxp_dither_lut_mode` {  
`kPXP_DitherLutOff` = 0U,  
`kPXP_DitherLutPreDither`,  
`kPXP_DitherLutPostDither` }
- PXP dither LUT mode.*
  - enum `_pxp_dither_matrix_size` {  
`kPXP_DitherMatrix8` = 1,  
`kPXP_DitherMatrix16` }
- PXP dither matrix size.*
  - enum {  
`kPXP_PorterDuffFactorOne` = 0U,  
`kPXP_PorterDuffFactorZero`,  
`kPXP_PorterDuffFactorStraight`,  
`kPXP_PorterDuffFactorInversed` }
- Porter Duff factor mode.*
  - enum {  
`kPXP_PorterDuffGlobalAlpha` = 0U,  
`kPXP_PorterDuffLocalAlpha`,  
`kPXP_PorterDuffScaledAlpha` }
- Porter Duff global alpha mode.*
  - enum {  
`kPXP_PorterDuffAlphaStraight` = 0U,  
`kPXP_PorterDuffAlphaInversed` }
- Porter Duff alpha mode.*
  - enum {  
`kPXP_PorterDuffColorStraight` = 0,  
`kPXP_PorterDuffColorInversed` = 1,  
`kPXP_PorterDuffColorNoAlpha` = 0,  
`kPXP_PorterDuffColorWithAlpha` = 1 }
- Porter Duff color mode.*
  - enum `pxp_porter_duff_blend_mode_t` {

```

kPXP_PorterDuffSrc = 0,
kPXP_PorterDuffAtop,
kPXP_PorterDuffOver,
kPXP_PorterDuffIn,
kPXP_PorterDuffOut,
kPXP_PorterDuffDst,
kPXP_PorterDuffDstAtop,
kPXP_PorterDuffDstOver,
kPXP_PorterDuffDstIn,
kPXP_PorterDuffDstOut,
kPXP_PorterDuffXor,
kPXP_PorterDuffClear }
 PXP Porter Duff blend mode.

```

## Driver version

- #define **FSL\_PXP\_DRIVER\_VERSION** ([MAKE\\_VERSION](#)(2, 2, 2))

## Initialization and deinitialization

- void [PXP\\_Init](#) (PXP\_Type \*base)  
*Initialize the PXP.*
- void [PXP\\_Deinit](#) (PXP\_Type \*base)  
*De-initialize the PXP.*
- void [PXP\\_Reset](#) (PXP\_Type \*base)  
*Reset the PXP.*

## Global operations

- static void [PXP\\_Start](#) (PXP\_Type \*base)  
*Start process.*
- static void [PXP\\_EnableLcdHandShake](#) (PXP\_Type \*base, bool enable)  
*Enable or disable LCD hand shake.*
- static void [PXP\\_EnableContinuousRun](#) (PXP\_Type \*base, bool enable)  
*Enable or disable continuous run.*
- static void [PXP\\_SetProcessBlockSize](#) (PXP\_Type \*base, [pxp\\_block\\_size\\_t](#) size)  
*Set the PXP processing block size.*

## Status

- static uint32\_t [PXP\\_GetStatusFlags](#) (PXP\_Type \*base)  
*Gets PXP status flags.*
- static void [PXP\\_ClearStatusFlags](#) (PXP\_Type \*base, uint32\_t statusMask)  
*Clears status flags with the provided mask.*
- static uint8\_t [PXP\\_GetAxiErrorId](#) (PXP\_Type \*base, uint8\_t axiIndex)  
*Gets the AXI ID of the failing bus operation.*

## Interrupts

- static void [PXP\\_EnableInterrupts](#) (PXP\_Type \*base, uint32\_t mask)

- Enables PXP interrupts according to the provided mask.
- static void [PXP\\_DisableInterrupts](#) (PXP\_Type \*base, uint32\_t mask)  
Disables PXP interrupts according to the provided mask.

## Alpha surface

- void [PXP\\_SetAlphaSurfaceBufferConfig](#) (PXP\_Type \*base, const [pxp\\_as\\_buffer\\_config\\_t](#) \*config)  
Set the alpha surface input buffer configuration.
- void [PXP\\_SetAlphaSurfaceBlendConfig](#) (PXP\_Type \*base, const [pxp\\_as\\_blend\\_config\\_t](#) \*config)  
Set the alpha surface blending configuration.
- void [PXP\\_SetAlphaSurfaceOverlayColorKey](#) (PXP\_Type \*base, uint32\_t colorKeyLow, uint32\_t colorKeyHigh)  
Set the alpha surface overlay color key.
- static void [PXP\\_EnableAlphaSurfaceOverlayColorKey](#) (PXP\_Type \*base, bool enable)  
Enable or disable the alpha surface color key.
- void [PXP\\_SetAlphaSurfacePosition](#) (PXP\_Type \*base, uint16\_t upperLeftX, uint16\_t upperLeftY, uint16\_t lowerRightX, uint16\_t lowerRightY)  
Set the alpha surface position in output buffer.

## Process surface

- static void [PXP\\_SetProcessSurfaceBackgroundColor](#) (PXP\_Type \*base, uint32\_t backGroundColor)  
Set the back ground color of PS.
- void [PXP\\_SetProcessSurfaceBufferConfig](#) (PXP\_Type \*base, const [pxp\\_ps\\_buffer\\_config\\_t](#) \*config)  
Set the process surface input buffer configuration.
- void [PXP\\_SetProcessSurfaceScaler](#) (PXP\_Type \*base, uint16\_t inputWidth, uint16\_t inputHeight, uint16\_t outputWidth, uint16\_t outputHeight)  
Set the process surface scaler configuration.
- void [PXP\\_SetProcessSurfacePosition](#) (PXP\_Type \*base, uint16\_t upperLeftX, uint16\_t upperLeftY, uint16\_t lowerRightX, uint16\_t lowerRightY)  
Set the process surface position in output buffer.
- void [PXP\\_SetProcessSurfaceColorKey](#) (PXP\_Type \*base, uint32\_t colorKeyLow, uint32\_t colorKeyHigh)  
Set the process surface color key.
- static void [PXP\\_SetProcessSurfaceYUVFormat](#) (PXP\_Type \*base, [pxp\\_ps\\_yuv\\_format\\_t](#) format)  
Set the process surface input pixel format YUV or YCbCr.

## Output buffer

- void [PXP\\_SetOutputBufferConfig](#) (PXP\_Type \*base, const [pxp\\_output\\_buffer\\_config\\_t](#) \*config)  
Set the PXP output buffer configuration.
- static void [PXP\\_SetOverwrittenAlphaValue](#) (PXP\_Type \*base, uint8\_t alpha)  
Set the global overwritten alpha value.
- static void [PXP\\_EnableOverWrittenAlpha](#) (PXP\_Type \*base, bool enable)  
Enable or disable the global overwritten alpha value.
- static void [PXP\\_SetRotateConfig](#) (PXP\_Type \*base, [pxp\\_rotate\\_position\\_t](#) position, [pxp\\_rotate\\_degree\\_t](#) degree, [pxp\\_flip\\_mode\\_t](#) flipMode)  
Set the rotation configuration.

## Command queue

- void [PXP\\_SetNextCommand](#) (PXP\_Type \*base, void \*commandAddr)  
*Set the next command.*
- static bool [PXP\\_IsNextCommandPending](#) (PXP\_Type \*base)  
*Check whether the next command is pending.*
- static void [PXP\\_CancelNextCommand](#) (PXP\_Type \*base)  
*Cancel command set by [PXP\\_SetNextCommand](#).*

## Color space conversion

- void [PXP\\_SetCsc1Mode](#) (PXP\_Type \*base, [pxp\\_csc1\\_mode\\_t](#) mode)  
*Set the CSC1 mode.*
- static void [PXP\\_EnableCsc1](#) (PXP\_Type \*base, bool enable)  
*Enable or disable the CSC1.*

## Porter Duff

- void [PXP\\_SetPorterDuffConfig](#) (PXP\_Type \*base, const [pxp\\_porter\\_duff\\_config\\_t](#) \*config)  
*Set the Porter Duff configuration.*
- [status\\_t](#) [PXP\\_GetPorterDuffConfig](#) ([pxp\\_porter\\_duff\\_blend\\_mode\\_t](#) mode, [pxp\\_porter\\_duff\\_config\\_t](#) \*config)  
*Get the Porter Duff configuration by blend mode.*

## Buffer copy

- [status\\_t](#) [PXP\\_StartPictureCopy](#) (PXP\_Type \*base, const [pxp\\_pic\\_copy\\_config\\_t](#) \*config)  
*Copy picture from one buffer to another buffer.*
- [status\\_t](#) [PXP\\_StartMemCopy](#) (PXP\_Type \*base, uint32\_t srcAddr, uint32\_t destAddr, uint32\_t size)  
*Copy continous memory.*

## 52.3 Data Structure Documentation

### 52.3.1 struct [pxp\\_output\\_buffer\\_config\\_t](#)

#### Data Fields

- [pxp\\_output\\_pixel\\_format\\_t](#) pixelFormat  
*Output buffer pixel format.*
- [pxp\\_interlaced\\_output\\_mode\\_t](#) interlacedMode  
*Interlaced output mode.*
- uint32\_t [buffer0Addr](#)  
*Output buffer 0 address.*
- uint32\_t [buffer1Addr](#)  
*Output buffer 1 address, used for UV data in YUV 2-plane mode, or field 1 in output interlaced mode.*
- uint16\_t [pitchBytes](#)  
*Number of bytes between two vertically adjacent pixels.*
- uint16\_t [width](#)  
*Pixels per line.*
- uint16\_t [height](#)  
*How many lines in output buffer.*

**Field Documentation**

- (1) `pxp_output_pixel_format_t` `pxp_output_buffer_config_t::pixelFormat`
- (2) `pxp_interlaced_output_mode_t` `pxp_output_buffer_config_t::interlacedMode`
- (3) `uint32_t` `pxp_output_buffer_config_t::buffer0Addr`
- (4) `uint32_t` `pxp_output_buffer_config_t::buffer1Addr`
- (5) `uint16_t` `pxp_output_buffer_config_t::pitchBytes`
- (6) `uint16_t` `pxp_output_buffer_config_t::width`
- (7) `uint16_t` `pxp_output_buffer_config_t::height`

**52.3.2 struct `pxp_ps_buffer_config_t`****Data Fields**

- `pxp_ps_pixel_format_t` `pixelFormat`  
*PS buffer pixel format.*
- `bool` `swapByte`  
*For each 16 bit word, set true to swap the two bytes.*
- `uint32_t` `bufferAddr`  
*Input buffer address for the first panel.*
- `uint32_t` `bufferAddrU`  
*Input buffer address for the second panel.*
- `uint32_t` `bufferAddrV`  
*Input buffer address for the third panel.*
- `uint16_t` `pitchBytes`  
*Number of bytes between two vertically adjacent pixels.*

**Field Documentation**

- (1) `pxp_ps_pixel_format_t` `pxp_ps_buffer_config_t::pixelFormat`
- (2) `bool` `pxp_ps_buffer_config_t::swapByte`
- (3) `uint32_t` `pxp_ps_buffer_config_t::bufferAddr`
- (4) `uint32_t` `pxp_ps_buffer_config_t::bufferAddrU`
- (5) `uint32_t` `pxp_ps_buffer_config_t::bufferAddrV`
- (6) `uint16_t` `pxp_ps_buffer_config_t::pitchBytes`

### 52.3.3 struct pxp\_as\_buffer\_config\_t

#### Data Fields

- [pxp\\_as\\_pixel\\_format\\_t pixelFormat](#)  
*AS buffer pixel format.*
- [uint32\\_t bufferAddr](#)  
*Input buffer address.*
- [uint16\\_t pitchBytes](#)  
*Number of bytes between two vertically adjacent pixels.*

#### Field Documentation

(1) [pxp\\_as\\_pixel\\_format\\_t pxp\\_as\\_buffer\\_config\\_t::pixelFormat](#)

(2) [uint32\\_t pxp\\_as\\_buffer\\_config\\_t::bufferAddr](#)

(3) [uint16\\_t pxp\\_as\\_buffer\\_config\\_t::pitchBytes](#)

### 52.3.4 struct pxp\_as\_blend\_config\_t

#### Data Fields

- [uint8\\_t alpha](#)  
*User defined alpha value, only used when [alphaMode](#) is [kPXP\\_AlphaOverride](#) or [kPXP\\_AlphaRop](#).*
- [bool invertAlpha](#)  
*Set true to invert the alpha.*
- [pxp\\_alpha\\_mode\\_t alphaMode](#)  
*Alpha mode.*
- [pxp\\_rop\\_mode\\_t ropMode](#)  
*ROP mode, only valid when [alphaMode](#) is [kPXP\\_AlphaRop](#).*

#### Field Documentation

(1) [uint8\\_t pxp\\_as\\_blend\\_config\\_t::alpha](#)

(2) [bool pxp\\_as\\_blend\\_config\\_t::invertAlpha](#)

(3) [pxp\\_alpha\\_mode\\_t pxp\\_as\\_blend\\_config\\_t::alphaMode](#)

(4) [pxp\\_rop\\_mode\\_t pxp\\_as\\_blend\\_config\\_t::ropMode](#)

### 52.3.5 struct pxp\_csc2\_config\_t

Converting from YUV/YCbCr color spaces to the RGB color space uses the following equation structure:

$$R = A1(Y+D1) + A2(U+D2) + A3(V+D3) \quad G = B1(Y+D1) + B2(U+D2) + B3(V+D3) \quad B = C1(Y+D1) + C2(U+D2) + C3(V+D3)$$



Converting from the RGB color space to YUV/YCbCr color spaces uses the following equation structure:

$$Y = A1 * R + A2 * G + A3 * B + D1 \quad U = B1 * R + B2 * G + B3 * B + D2 \quad V = C1 * R + C2 * G + C3 * B + D3$$

## Data Fields

- [pxp\\_csc2\\_mode\\_t mode](#)  
*Conversion mode.*
- float [A1](#)  
*A1.*
- float [A2](#)  
*A2.*
- float [A3](#)  
*A3.*
- float [B1](#)  
*B1.*
- float [B2](#)  
*B2.*
- float [B3](#)  
*B3.*
- float [C1](#)  
*C1.*
- float [C2](#)  
*C2.*
- float [C3](#)  
*C3.*
- int16\_t [D1](#)  
*D1.*
- int16\_t [D2](#)  
*D2.*
- int16\_t [D3](#)  
*D3.*

## Field Documentation

- (1) `pxp_csc2_mode_t pxp_csc2_config_t::mode`
- (2) `float pxp_csc2_config_t::A1`
- (3) `float pxp_csc2_config_t::A2`
- (4) `float pxp_csc2_config_t::A3`
- (5) `float pxp_csc2_config_t::B1`
- (6) `float pxp_csc2_config_t::B2`
- (7) `float pxp_csc2_config_t::B3`
- (8) `float pxp_csc2_config_t::C1`

(9) float pxp\_csc2\_config\_t::C2

(10) float pxp\_csc2\_config\_t::C3

(11) int16\_t pxp\_csc2\_config\_t::D1

(12) int16\_t pxp\_csc2\_config\_t::D2

(13) int16\_t pxp\_csc2\_config\_t::D3

### 52.3.6 struct pxp\_dither\_final\_lut\_data\_t

#### Data Fields

- uint32\_t [data\\_3\\_0](#)  
*Data 3 to data 0.*
- uint32\_t [data\\_7\\_4](#)  
*Data 7 to data 4.*
- uint32\_t [data\\_11\\_8](#)  
*Data 11 to data 8.*
- uint32\_t [data\\_15\\_12](#)  
*Data 15 to data 12.*

#### Field Documentation

(1) uint32\_t pxp\_dither\_final\_lut\_data\_t::data\_3\_0

Data 0 is the least significant byte.

(2) uint32\_t pxp\_dither\_final\_lut\_data\_t::data\_7\_4

Data 4 is the least significant byte.

(3) uint32\_t pxp\_dither\_final\_lut\_data\_t::data\_11\_8

Data 8 is the least significant byte.

(4) uint32\_t pxp\_dither\_final\_lut\_data\_t::data\_15\_12

Data 12 is the least significant byte.

### 52.3.7 struct pxp\_dither\_config\_t

#### Data Fields

- uint32\_t [enableDither0](#): 1  
*Enable dither engine 0 or not, set 1 to enable, 0 to disable.*
- uint32\_t [enableDither1](#): 1

- `uint32_t enableDither2`: 1  
*Enable dither engine 1 or not, set 1 to enable, 0 to disable.*
- `uint32_t ditherMode0`: 3  
*Enable dither engine 2 or not, set 1 to enable, 0 to disable.*
- `uint32_t ditherMode1`: 3  
*Dither mode for dither engine 0.*
- `uint32_t ditherMode2`: 3  
*Dither mode for dither engine 1.*
- `uint32_t quantBitNum`: 3  
*Dither mode for dither engine 2.*
- `uint32_t lutMode`: 2  
*Number of bits quantize down to, the valid value is 1~7.*
- `uint32_t idxMatrixSize0`: 2  
*How to use the memory LUT, see [\\_pxp\\_dither\\_lut\\_mode](#).*
- `uint32_t idxMatrixSize1`: 2  
*Size of index matrix used for dither for dither engine 0, see [\\_pxp\\_dither\\_matrix\\_size](#).*
- `uint32_t idxMatrixSize2`: 2  
*Size of index matrix used for dither for dither engine 1, see [\\_pxp\\_dither\\_matrix\\_size](#).*
- `uint32_t enableFinalLut`: 1  
*Size of index matrix used for dither for dither engine 2, see [\\_pxp\\_dither\\_matrix\\_size](#).*
- `uint32_t enableFinalLut`: 1  
*Enable the final LUT, set 1 to enable, 0 to disable.*

### Field Documentation

(1) `uint32_t pxp_dither_config_t::enableDither0`

(2) `uint32_t pxp_dither_config_t::enableDither1`

(3) `uint32_t pxp_dither_config_t::enableDither2`

(4) `uint32_t pxp_dither_config_t::ditherMode0`

See [\\_pxp\\_dither\\_mode](#).

(5) `uint32_t pxp_dither_config_t::ditherMode1`

See [\\_pxp\\_dither\\_mode](#).

(6) `uint32_t pxp_dither_config_t::ditherMode2`

See [\\_pxp\\_dither\\_mode](#).

(7) `uint32_t pxp_dither_config_t::quantBitNum`

(8) `uint32_t pxp_dither_config_t::lutMode`

This must be set to [kPXP\\_DitherLutOff](#) if any dither engine uses [kPXP\\_DitherOrdered](#) mode.

(9) `uint32_t pxp_dither_config_t::idxMatrixSize0`

(10) `uint32_t pxp_dither_config_t::idxMatrixSize1`

(11) `uint32_t pxp_dither_config_t::idxMatrixSize2`

(12) `uint32_t pxp_dither_config_t::enableFinalLut`

### 52.3.8 struct `pxp_porter_duff_config_t`

#### Data Fields

- `uint32_t enable`: 1  
*Enable or disable Porter Duff.*
- `uint32_t srcFactorMode`: 2  
*Source layer (or AS, s1) factor mode, see [pxp\\_porter\\_duff\\_factor\\_mode](#).*
- `uint32_t dstGlobalAlphaMode`: 2  
*Destination layer (or PS, s0) global alpha mode, see [pxp\\_porter\\_duff\\_global\\_alpha\\_mode](#).*
- `uint32_t dstAlphaMode`: 1  
*Destination layer (or PS, s0) alpha mode, see [pxp\\_porter\\_duff\\_alpha\\_mode](#).*
- `uint32_t dstColorMode`: 1  
*Destination layer (or PS, s0) color mode, see [pxp\\_porter\\_duff\\_color\\_mode](#).*
- `uint32_t dstFactorMode`: 2  
*Destination layer (or PS, s0) factor mode, see [pxp\\_porter\\_duff\\_factor\\_mode](#).*
- `uint32_t srcGlobalAlphaMode`: 2  
*Source layer (or AS, s1) global alpha mode, see [pxp\\_porter\\_duff\\_global\\_alpha\\_mode](#).*
- `uint32_t srcAlphaMode`: 1  
*Source layer (or AS, s1) alpha mode, see [pxp\\_porter\\_duff\\_alpha\\_mode](#).*
- `uint32_t srcColorMode`: 1  
*Source layer (or AS, s1) color mode, see [pxp\\_porter\\_duff\\_color\\_mode](#).*
- `uint32_t dstGlobalAlpha`: 8  
*Destination layer (or PS, s0) global alpha value, 0~255.*
- `uint32_t srcGlobalAlpha`: 8  
*Source layer (or AS, s1) global alpha value, 0~255.*

#### Field Documentation

(1) `uint32_t pxp_porter_duff_config_t::enable`

(2) `uint32_t pxp_porter_duff_config_t::srcFactorMode`

(3) `uint32_t pxp_porter_duff_config_t::dstGlobalAlphaMode`

(4) `uint32_t pxp_porter_duff_config_t::dstAlphaMode`

(5) `uint32_t pxp_porter_duff_config_t::dstColorMode`

(6) `uint32_t pxp_porter_duff_config_t::dstFactorMode`

(7) `uint32_t pxp_porter_duff_config_t::srcGlobalAlphaMode`

(8) `uint32_t pxp_porter_duff_config_t::srcAlphaMode`

(9) `uint32_t pxp_porter_duff_config_t::srcColorMode`

(10) `uint32_t pxp_porter_duff_config_t::dstGlobalAlpha`

(11) `uint32_t pxp_porter_duff_config_t::srcGlobalAlpha`

### 52.3.9 struct `pxp_pic_copy_config_t`

Note: don't change the enum item value

#### Data Fields

- `uint32_t srcPicBaseAddr`  
*Source picture base address.*
- `uint16_t srcPitchBytes`  
*Pitch of the source buffer.*
- `uint16_t srcOffsetX`  
*Copy position in source picture.*
- `uint16_t srcOffsetY`  
*Copy position in source picture.*
- `uint32_t destPicBaseAddr`  
*Destination picture base address.*
- `uint16_t destPitchBytes`  
*Pitch of the destination buffer.*
- `uint16_t destOffsetX`  
*Copy position in destination picture.*
- `uint16_t destOffsetY`  
*Copy position in destination picture.*
- `uint16_t width`  
*Pixel number each line to copy.*
- `uint16_t height`  
*Lines to copy.*
- `pxp_as_pixel_format_t pixelFormat`  
*Buffer pixel format.*

#### Field Documentation

(1) `uint32_t pxp_pic_copy_config_t::srcPicBaseAddr`

(2) `uint16_t pxp_pic_copy_config_t::srcPitchBytes`

(3) `uint16_t pxp_pic_copy_config_t::srcOffsetX`

(4) `uint16_t pxp_pic_copy_config_t::srcOffsetY`

(5) `uint32_t pxp_pic_copy_config_t::destPicBaseAddr`

(6) `uint16_t pxp_pic_copy_config_t::destPitchBytes`

(7) `uint16_t pxp_pic_copy_config_t::destOffsetX`

- (8) `uint16_t pxp_pic_copy_config_t::destOffsetY`
- (9) `uint16_t pxp_pic_copy_config_t::width`
- (10) `uint16_t pxp_pic_copy_config_t::height`
- (11) `pxp_as_pixel_format_t pxp_pic_copy_config_t::pixelFormat`

## 52.4 Enumeration Type Documentation

### 52.4.1 `enum _pxp_interrupt_enable`

Enumerator

***kPXP\_CommandLoadInterruptEnable*** Interrupt to show that the command set by [PXP\\_SetNextCommand](#) has been loaded.

***kPXP\_CompleteInterruptEnable*** PXP process completed.

### 52.4.2 `enum _pxp_flags`

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

***kPXP\_CommandLoadFlag*** The command set by [PXP\\_SetNextCommand](#) has been loaded, could set new command.

***kPXP\_CompleteFlag*** PXP process completed.

***kPXP\_Axi0ReadErrorFlag*** PXP encountered an AXI read error and processing has been terminated.

***kPXP\_Axi0WriteErrorFlag*** PXP encountered an AXI write error and processing has been terminated.

### 52.4.3 `enum pxp_flip_mode_t`

Enumerator

***kPXP\_FlipDisable*** Flip disable.

***kPXP\_FlipHorizontal*** Horizontal flip.

***kPXP\_FlipVertical*** Vertical flip.

***kPXP\_FlipBoth*** Flip both directions.

### 52.4.4 enum pxp\_rotate\_position\_t

Enumerator

***kPXP\_RotateOutputBuffer*** Rotate the output buffer.

***kPXP\_RotateProcessSurface*** Rotate the process surface.

### 52.4.5 enum pxp\_rotate\_degree\_t

Enumerator

***kPXP\_Rotate0*** Clock wise rotate 0 deg.

***kPXP\_Rotate90*** Clock wise rotate 90 deg.

***kPXP\_Rotate180*** Clock wise rotate 180 deg.

***kPXP\_Rotate270*** Clock wise rotate 270 deg.

### 52.4.6 enum pxp\_interlaced\_output\_mode\_t

Enumerator

***kPXP\_OutputProgressive*** All data written in progressive format to output buffer 0.

***kPXP\_OutputField0*** Only write field 0 data to output buffer 0.

***kPXP\_OutputField1*** Only write field 1 data to output buffer 0.

***kPXP\_OutputInterlaced*** Field 0 write to buffer 0, field 1 write to buffer 1.

### 52.4.7 enum pxp\_output\_pixel\_format\_t

Enumerator

***kPXP\_OutputPixelFormatARGB8888*** 32-bit pixels with alpha.

***kPXP\_OutputPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)

***kPXP\_OutputPixelFormatRGB888P*** 24-bit pixels without alpha (packed 24-bit format)

***kPXP\_OutputPixelFormatARGB1555*** 16-bit pixels with alpha.

***kPXP\_OutputPixelFormatARGB4444*** 16-bit pixels with alpha.

***kPXP\_OutputPixelFormatRGB555*** 16-bit pixels without alpha.

***kPXP\_OutputPixelFormatRGB444*** 16-bit pixels without alpha.

***kPXP\_OutputPixelFormatRGB565*** 16-bit pixels without alpha.

***kPXP\_OutputPixelFormatYUVIP444*** 32-bit pixels (1-plane XYUV unpacked).

***kPXP\_OutputPixelFormatUYVYIP422*** 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)

***kPXP\_OutputPixelFormatVYUYIP422*** 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)

***kPXP\_OutputPixelFormatY8*** 8-bit monochrome pixels (1-plane Y luma output)

***kPXP\_OutputPixelFormatY4*** 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)

***kPXP\_OutputPixelFormatYUV2P422*** 16-bit pixels (2-plane UV interleaved bytes)  
***kPXP\_OutputPixelFormatYUV2P420*** 16-bit pixels (2-plane UV)  
***kPXP\_OutputPixelFormatYVU2P422*** 16-bit pixels (2-plane VU interleaved bytes)  
***kPXP\_OutputPixelFormatYVU2P420*** 16-bit pixels (2-plane VU)

#### 52.4.8 enum pxp\_ps\_pixel\_format\_t

Enumerator

***kPXP\_PsPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)  
***kPXP\_PsPixelFormatRGB555*** 16-bit pixels without alpha.  
***kPXP\_PsPixelFormatRGB444*** 16-bit pixels without alpha.  
***kPXP\_PsPixelFormatRGB565*** 16-bit pixels without alpha.  
***kPXP\_PsPixelFormatYUV1P444*** 32-bit pixels (1-plane XYUV unpacked).  
***kPXP\_PsPixelFormatUYVY1P422*** 16-bit pixels (1-plane U0,Y0,V0,Y1 interleaved bytes)  
***kPXP\_PsPixelFormatVYUY1P422*** 16-bit pixels (1-plane V0,Y0,U0,Y1 interleaved bytes)  
***kPXP\_PsPixelFormatY8*** 8-bit monochrome pixels (1-plane Y luma output)  
***kPXP\_PsPixelFormatY4*** 4-bit monochrome pixels (1-plane Y luma, 4 bit truncation)  
***kPXP\_PsPixelFormatYUV2P422*** 16-bit pixels (2-plane UV interleaved bytes)  
***kPXP\_PsPixelFormatYUV2P420*** 16-bit pixels (2-plane UV)  
***kPXP\_PsPixelFormatYVU2P422*** 16-bit pixels (2-plane VU interleaved bytes)  
***kPXP\_PsPixelFormatYVU2P420*** 16-bit pixels (2-plane VU)  
***kPXP\_PsPixelFormatYVU422*** 16-bit pixels (3-plane)  
***kPXP\_PsPixelFormatYVU420*** 16-bit pixels (3-plane)

#### 52.4.9 enum pxp\_ps\_yuv\_format\_t

Enumerator

***kPXP\_PsYUVFormatYUV*** YUV format.  
***kPXP\_PsYUVFormatYCbCr*** YCbCr format.

#### 52.4.10 enum pxp\_as\_pixel\_format\_t

Enumerator

***kPXP\_AsPixelFormatARGB8888*** 32-bit pixels with alpha.  
***kPXP\_AsPixelFormatRGB888*** 32-bit pixels without alpha (unpacked 24-bit format)  
***kPXP\_AsPixelFormatARGB1555*** 16-bit pixels with alpha.  
***kPXP\_AsPixelFormatARGB4444*** 16-bit pixels with alpha.  
***kPXP\_AsPixelFormatRGB555*** 16-bit pixels without alpha.



***kPXP\_AsPixelFormatRGB444*** 16-bit pixels without alpha.

***kPXP\_AsPixelFormatRGB565*** 16-bit pixels without alpha.

### 52.4.11 enum pxp\_alpha\_mode\_t

Enumerator

***kPXP\_AlphaEmbedded*** The alpha surface pixel alpha value will be used for blend.

***kPXP\_AlphaOverride*** The user defined alpha value will be used for blend directly.

***kPXP\_AlphaMultiply*** The alpha surface pixel alpha value scaled the user defined alpha value will be used for blend, for example, pixel alpha set to 200, user defined alpha set to 100, then the result alpha is  $200 * 100 / 255$ .

***kPXP\_AlphaRop*** Raster operation.

### 52.4.12 enum pxp\_rop\_mode\_t

Explanation:

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

Enumerator

***kPXP\_RopMaskAs*** AS AND PS.

***kPXP\_RopMaskNotAs*** nAS AND PS.

***kPXP\_RopMaskAsNot*** AS AND nPS.

***kPXP\_RopMergeAs*** AS OR PS.

***kPXP\_RopMergeNotAs*** nAS OR PS.

***kPXP\_RopMergeAsNot*** AS OR nPS.

***kPXP\_RopNotCopyAs*** nAS.

***kPXP\_RopNot*** nPS.

***kPXP\_RopNotMaskAs*** AS NAND PS.

***kPXP\_RopNotMergeAs*** AS NOR PS.

***kPXP\_RopXorAs*** AS XOR PS.

***kPXP\_RopNotXorAs*** AS XNOR PS.

### 52.4.13 enum pxp\_block\_size\_t

Enumerator

***kPXP\_BlockSize8*** Process 8x8 pixel blocks.

***kPXP\_BlockSize16*** Process 16x16 pixel blocks.

#### 52.4.14 enum pxp\_csc1\_mode\_t

Enumerator

***kPXP\_Csc1YUV2RGB*** YUV to RGB.

***kPXP\_Csc1YCbCr2RGB*** YCbCr to RGB.

#### 52.4.15 enum pxp\_csc2\_mode\_t

Enumerator

***kPXP\_Csc2YUV2RGB*** YUV to RGB.

***kPXP\_Csc2YCbCr2RGB*** YCbCr to RGB.

***kPXP\_Csc2RGB2YUV*** RGB to YUV.

***kPXP\_Csc2RGB2YCbCr*** RGB to YCbCr.

#### 52.4.16 enum pxp\_ram\_t

Enumerator

***kPXP\_RamDither0Lut*** Dither 0 LUT memory.

***kPXP\_RamDither1Lut*** Dither 1 LUT memory.

***kPXP\_RamDither2Lut*** Dither 2 LUT memory.

#### 52.4.17 enum \_pxp\_dither\_mode

Enumerator

***kPXP\_DitherPassThrough*** Pass through, no dither.

***kPXP\_DitherOrdered*** Ordered dither.

***kPXP\_DitherQuantOnly*** No dithering, only quantization.

#### 52.4.18 enum \_pxp\_dither\_lut\_mode

Enumerator

***kPXP\_DitherLutOff*** The LUT memory is not used for LUT, could be used as ordered dither index matrix.

***kPXP\_DitherLutPreDither*** Use LUT at the pre-dither stage, The pre-dither LUT could only be used in Floyd mode or Atkinson mode, which are not supported by current PXP module.

***kPXP\_DitherLutPostDither*** Use LUT at the post-dither stage.

## 52.4.19 enum \_pxp\_dither\_matrix\_size

Enumerator

***kPXP\_DitherMatrix8*** The dither index matrix is 8x8.

***kPXP\_DitherMatrix16*** The dither index matrix is 16x16.

## 52.4.20 anonymous enum

Enumerator

***kPXP\_PorterDuffFactorOne*** Use 1.

***kPXP\_PorterDuffFactorZero*** Use 0.

***kPXP\_PorterDuffFactorStraight*** Use straight alpha.

***kPXP\_PorterDuffFactorInversed*** Use inversed alpha.

## 52.4.21 anonymous enum

Enumerator

***kPXP\_PorterDuffGlobalAlpha*** Use global alpha.

***kPXP\_PorterDuffLocalAlpha*** Use local alpha in each pixel.

***kPXP\_PorterDuffScaledAlpha*** Use global alpha \* local alpha.

## 52.4.22 anonymous enum

Enumerator

***kPXP\_PorterDuffAlphaStraight*** Use straight alpha, s0\_alpha' = s0\_alpha.

***kPXP\_PorterDuffAlphaInversed*** Use inversed alpha, s0\_alpha' = 0xFF - s0\_alpha.

## 52.4.23 anonymous enum

Enumerator

***kPXP\_PorterDuffColorStraight*** **Deprecated** Use kPXP\_PorterDuffColorNoAlpha.

***kPXP\_PorterDuffColorInversed*** **Deprecated** Use kPXP\_PorterDuffColorWithAlpha.

***kPXP\_PorterDuffColorNoAlpha*** s0\_pixel' = s0\_pixel.

*kPXP\_PorterDuffColorWithAlpha*  $s0\_pixel' = s0\_pixel * s0\_alpha$ ".

#### 52.4.24 enum pxp\_porter\_duff\_blend\_mode\_t

Note: don't change the enum item value

Enumerator

*kPXP\_PorterDuffSrc* Source Only.  
*kPXP\_PorterDuffAtop* Source Atop.  
*kPXP\_PorterDuffOver* Source Over.  
*kPXP\_PorterDuffIn* Source In.  
*kPXP\_PorterDuffOut* Source Out.  
*kPXP\_PorterDuffDst* Destination Only.  
*kPXP\_PorterDuffDstAtop* Destination Atop.  
*kPXP\_PorterDuffDstOver* Destination Over.  
*kPXP\_PorterDuffDstIn* Destination In.  
*kPXP\_PorterDuffDstOut* Destination Out.  
*kPXP\_PorterDuffXor* XOR.  
*kPXP\_PorterDuffClear* Clear.

### 52.5 Function Documentation

#### 52.5.1 void PXP\_Init ( PXP\_Type \* *base* )

This function enables the PXP peripheral clock, and resets the PXP registers to default status.

Parameters

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

#### 52.5.2 void PXP\_Deinit ( PXP\_Type \* *base* )

This function disables the PXP peripheral clock.

Parameters

---

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

### 52.5.3 void PXP\_Reset ( PXP\_Type \* *base* )

This function resets the PXP peripheral registers to default status.

Parameters

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

### 52.5.4 static void PXP\_Start ( PXP\_Type \* *base* ) [inline], [static]

Start PXP process using current configuration.

Parameters

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

### 52.5.5 static void PXP\_EnableLcdHandShake ( PXP\_Type \* *base*, bool *enable* ) [inline], [static]

Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

### 52.5.6 static void PXP\_EnableContinuousRun ( PXP\_Type \* *base*, bool *enable* ) [inline], [static]

If continuous run not enabled, [PXP\\_Start](#) starts the PXP process. When completed, PXP enters idle mode and flag [kPXP\\_CompleteFlag](#) asserts.

If continuous run enabled, the PXP will repeat based on the current configuration register settings.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

### 52.5.7 static void PXP\_SetProcessBlockSize ( PXP\_Type \* *base*, pxp\_block\_size\_t *size* ) [inline], [static]

This function chooses the pixel block size that PXP using during process. Larger block size means better performance, but be careful that when PXP is rotating, the output must be divisible by the block size selected.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | PXP peripheral base address. |
| <i>size</i> | The pixel block size.        |

### 52.5.8 static uint32\_t PXP\_GetStatusFlags ( PXP\_Type \* *base* ) [inline], [static]

This function gets all PXP status flags. The flags are returned as the logical OR value of the enumerators [\\_pxp\\_flags](#). To check a specific status, compare the return value with enumerators in [\\_pxp\\_flags](#). For example, to check whether the PXP has completed process, use like this:

```
if (kPXP_CompleteFlag & PXP_GetStatusFlags(PXP))
{
 ...
}
```

## Parameters

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

## Returns

PXP status flags which are OR'ed by the enumerators in the [\\_pxp\\_flags](#).

### 52.5.9 static void PXP\_ClearStatusFlags ( PXP\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

This function clears PXP status flags with a provided mask.

## Parameters

|                   |                                                                                        |
|-------------------|----------------------------------------------------------------------------------------|
| <i>base</i>       | PXP peripheral base address.                                                           |
| <i>statusMask</i> | The status flags to be cleared; it is logical OR value of <a href="#">_pxp_flags</a> . |

### 52.5.10 static uint8\_t PXP\_GetAxiErrorId ( PXP\_Type \* *base*, uint8\_t *axiIndex* ) [inline], [static]

## Parameters

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>     | PXP peripheral base address.                                                                     |
| <i>axiIndex</i> | Whitch AXI to get <ul style="list-style-type: none"> <li>• 0: AXI0</li> <li>• 1: AXI1</li> </ul> |

## Returns

The AXI ID of the failing bus operation.

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

This function enables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_pxp\\_interrupt\\_enable](#). For example, to enable PXP process complete interrupt and command loaded interrupt, do the following.

```
PXP_EnableInterrupts(PXP, kPXP_CommandLoadInterruptEnable
| kPXP_CompleteInterruptEnable);
```

## Parameters

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>base</i> | PXP peripheral base address.                                                    |
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_pxp_interrupt_enable</a> . |

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

This function disables the PXP interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_pxp\\_interrupt\\_enable](#).

## Parameters

|             |                                                                                  |
|-------------|----------------------------------------------------------------------------------|
| <i>base</i> | PXP peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_pxp_interrupt_enable</a> . |

### 52.5.13 void PXP\_SetAlphaSurfaceBufferConfig ( PXP\_Type \* *base*, const pxp\_as\_buffer\_config\_t \* *config* )

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | PXP peripheral base address.  |
| <i>config</i> | Pointer to the configuration. |

### 52.5.14 void PXP\_SetAlphaSurfaceBlendConfig ( PXP\_Type \* *base*, const pxp\_as\_blend\_config\_t \* *config* )

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | PXP peripheral base address.            |
| <i>config</i> | Pointer to the configuration structure. |

### 52.5.15 void PXP\_SetAlphaSurfaceOverlayColorKey ( PXP\_Type \* *base*, uint32\_t *colorKeyLow*, uint32\_t *colorKeyHigh* )

If a pixel in the current overlay image with a color that falls in the range from the `colorKeyLow` to `colorKeyHigh` range, it will use the process surface pixel value for that location. If no PS image is present or if the PS image also matches its colorkey range, the PS background color is used.

## Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>colorKeyLow</i> | Color key low range.         |



|                     |                       |
|---------------------|-----------------------|
| <i>colorKeyHigh</i> | Color key high range. |
|---------------------|-----------------------|

## Note

Colorkey operations are higher priority than alpha or ROP operations

**52.5.16 static void PXP\_EnableAlphaSurfaceOverlayColorKey ( PXP\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

**52.5.17 void PXP\_SetAlphaSurfacePosition ( PXP\_Type \* *base*, uint16\_t *upperLeftX*, uint16\_t *upperLeftY*, uint16\_t *lowerRightX*, uint16\_t *lowerRightY* )**

## Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>upperLeftX</i>  | X of the upper left corner.  |
| <i>upperLeftY</i>  | Y of the upper left corner.  |
| <i>lowerRightX</i> | X of the lower right corner. |
| <i>lowerRightY</i> | Y of the lower right corner. |

**52.5.18 static void PXP\_SetProcessSurfaceBackgroundColor ( PXP\_Type \* *base*, uint32\_t *backgroundColor* ) [inline], [static]**

## Parameters

|                         |                                      |
|-------------------------|--------------------------------------|
| <i>base</i>             | PXP peripheral base address.         |
| <i>backGround-Color</i> | Pixel value of the background color. |

**52.5.19** void PXP\_SetProcessSurfaceBufferConfig ( PXP\_Type \* *base*, const  
pxp\_ps\_buffer\_config\_t \* *config* )

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | PXP peripheral base address.  |
| <i>config</i> | Pointer to the configuration. |

**52.5.20 void PXP\_SetProcessSurfaceScaler ( PXP\_Type \* *base*, uint16\_t *inputWidth*, uint16\_t *inputHeight*, uint16\_t *outputWidth*, uint16\_t *outputHeight* )**

The valid down scale fact is  $1/(2^{12}) \sim 16$ .

## Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | PXP peripheral base address. |
| <i>inputWidth</i>   | Input image width.           |
| <i>inputHeight</i>  | Input image height.          |
| <i>outputWidth</i>  | Output image width.          |
| <i>outputHeight</i> | Output image height.         |

**52.5.21 void PXP\_SetProcessSurfacePosition ( PXP\_Type \* *base*, uint16\_t *upperLeftX*, uint16\_t *upperLeftY*, uint16\_t *lowerRightX*, uint16\_t *lowerRightY* )**

## Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>upperLeftX</i>  | X of the upper left corner.  |
| <i>upperLeftY</i>  | Y of the upper left corner.  |
| <i>lowerRightX</i> | X of the lower right corner. |
| <i>lowerRightY</i> | Y of the lower right corner. |

**52.5.22 void PXP\_SetProcessSurfaceColorKey ( PXP\_Type \* *base*, uint32\_t *colorKeyLow*, uint32\_t *colorKeyHigh* )**

If the PS image matches colorkey range, the PS background color is output. Set *colorKeyLow* to 0xF-FFFFFF and *colorKeyHigh* to 0 will disable the colorkeying.

## Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | PXP peripheral base address. |
| <i>colorKeyLow</i>  | Color key low range.         |
| <i>colorKeyHigh</i> | Color key high range.        |

### 52.5.23 static void PXP\_SetProcessSurfaceYUVFormat ( PXP\_Type \* *base*, pxp\_ps\_yuv\_format\_t *format* ) [inline], [static]

If process surface input pixel format is YUV and CSC1 is not enabled, in other words, the process surface output pixel format is also YUV, then this function should be called to set whether input pixel format is YUV or YCbCr.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | PXP peripheral base address. |
| <i>format</i> | The YUV format.              |

### 52.5.24 void PXP\_SetOutputBufferConfig ( PXP\_Type \* *base*, const pxp\_output\_buffer\_config\_t \* *config* )

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | PXP peripheral base address.  |
| <i>config</i> | Pointer to the configuration. |

### 52.5.25 static void PXP\_SetOverwrittenAlphaValue ( PXP\_Type \* *base*, uint8\_t *alpha* ) [inline], [static]

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

## Parameters

|              |                              |
|--------------|------------------------------|
| <i>base</i>  | PXP peripheral base address. |
| <i>alpha</i> | The alpha value.             |

**52.5.26** `static void PXP_EnableOverWrittenAlpha ( PXP_Type * base, bool enable  
 ) [inline], [static]`

If global overwritten alpha is enabled, the alpha component in output buffer pixels will be overwritten, otherwise the computed alpha value is used.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

### 52.5.27 static void PXP\_SetRotateConfig ( PXP\_Type \* *base*, pxp\_rotate\_position\_t *position*, pxp\_rotate\_degree\_t *degree*, pxp\_flip\_mode\_t *flipMode* ) [inline], [static]

The PXP could rotate the process surface or the output buffer. There are two PXP versions:

- Version 1: Only has one rotate sub module, the output buffer and process surface share the same rotate sub module, which means the process surface and output buffer could not be rotate at the same time. When pass in [kPXP\\_RotateOutputBuffer](#), the process surface could not use the rotate, Also when pass in [kPXP\\_RotateProcessSurface](#), output buffer could not use the rotate.
- Version 2: Has two separate rotate sub modules, the output buffer and process surface could configure the rotation independently.

Upper layer could use the macro PXP\_SHARE\_ROTATE to check which version is. PXP\_SHARE\_ROTATE=1 means version 1.

## Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>base</i>     | PXP peripheral base address.             |
| <i>position</i> | Rotate process surface or output buffer. |
| <i>degree</i>   | Rotate degree.                           |
| <i>flipMode</i> | Flip mode.                               |

## Note

This function is different depends on the macro PXP\_SHARE\_ROTATE.

### 52.5.28 void PXP\_SetNextCommand ( PXP\_Type \* *base*, void \* *commandAddr* )

The PXP supports a primitive ability to queue up one operation while the current operation is running. Workflow:

1. Prepare the PXP register values except STAT, CSCCOEFn, NEXT in the memory in the order they appear in the register map.
2. Call this function sets the new operation to PXP.
3. There are two methods to check whether the PXP has loaded the new operation. The first method is using [PXP\\_IsNextCommandPending](#). If there is new operation not loaded by the PXP, this function

returns true. The second method is checking the flag `kPXP_CommandLoadFlag`, if command loaded, this flag asserts. User could enable interrupt `kPXP_CommandLoadInterruptEnable` to get the loaded signal in interrupt way.

4. When command loaded by PXP, a new command could be set using this function.

```
uint32_t pxp_command1[48];
uint32_t pxp_command2[48];

pxp_command1[0] = ...;
pxp_command1[1] = ...;
...
pxp_command2[0] = ...;
pxp_command2[1] = ...;
...

while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command1);

while (PXP_IsNextCommandPending(PXP))
{
}

PXP_SetNextCommand(PXP, pxp_command2);
```

#### Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | PXP peripheral base address. |
| <i>commandAddr</i> | Address of the new command.  |

### 52.5.29 static bool PXP\_IsNextCommandPending ( PXP\_Type \* *base* ) [inline], [static]

#### Parameters

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

#### Returns

True is pending, false is not.

### 52.5.30 static void PXP\_CancelNextCommand ( PXP\_Type \* *base* ) [inline], [static]

## Parameters

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

**52.5.31 void PXP\_SetCsc1Mode ( PXP\_Type \* *base*, pxp\_csc1\_mode\_t *mode* )**

The CSC1 module receives scaled YUV/YCbCr444 pixels from the scale engine and converts the pixels to the RGB888 color space. It could only be used by process surface.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | PXP peripheral base address. |
| <i>mode</i> | The conversion mode.         |

**52.5.32 static void PXP\_EnableCsc1 ( PXP\_Type \* *base*, bool *enable* )  
[inline], [static]**

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | PXP peripheral base address.      |
| <i>enable</i> | True to enable, false to disable. |

**52.5.33 void PXP\_SetPorterDuffConfig ( PXP\_Type \* *base*, const  
pxp\_porter\_duff\_config\_t \* *config* )**

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | PXP peripheral base address.  |
| <i>config</i> | Pointer to the configuration. |

**52.5.34 status\_t PXP\_GetPorterDuffConfig ( pxp\_porter\_duff\_blend\_mode\_t *mode*,  
pxp\_porter\_duff\_config\_t \* *config* )**

The FactorMode are selected based on blend mode, the AlphaMode are set to [kPXP\\_PorterDuffAlphaStraight](#), the ColorMode are set to [kPXP\\_PorterDuffColorWithAlpha](#), the GlobalAlphaMode are set to [kPXP\\_PorterDuffLocalAlpha](#). These values could be modified after calling this function.



## Parameters

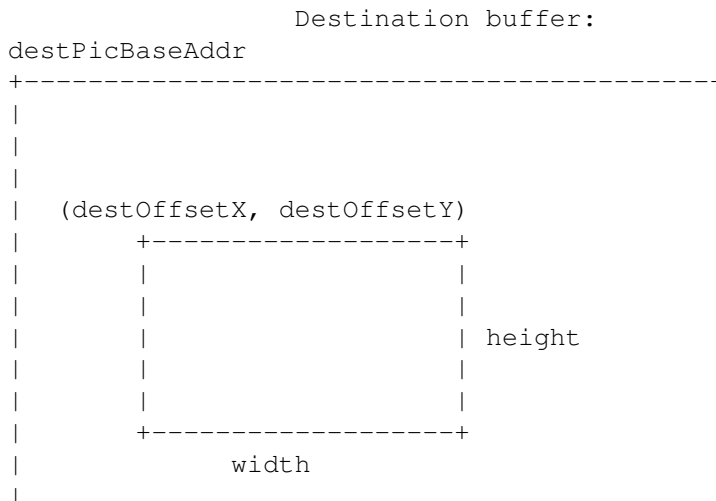
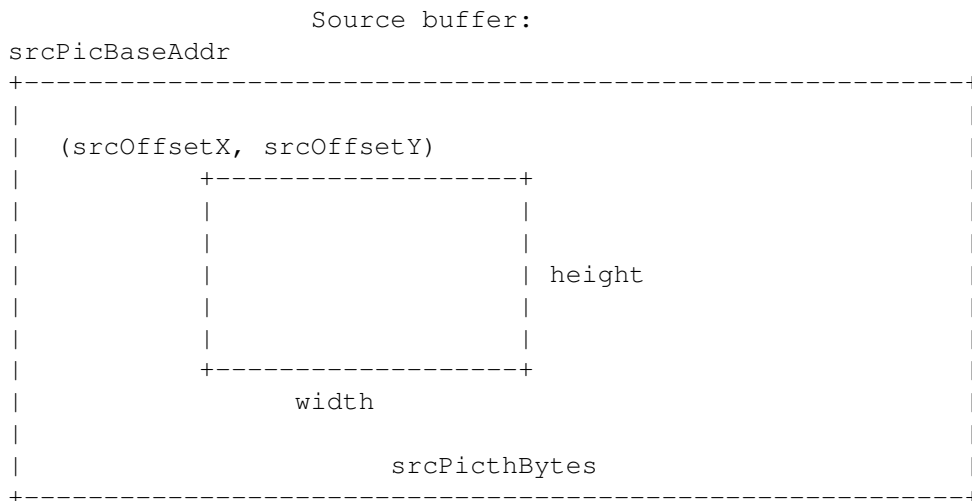
|               |                               |
|---------------|-------------------------------|
| <i>mode</i>   | The blend mode.               |
| <i>config</i> | Pointer to the configuration. |

## Return values

|                                |                                     |
|--------------------------------|-------------------------------------|
| <i>kStatus_Success</i>         | Successfully get the configuratoin. |
| <i>kStatus_InvalidArgument</i> | The blend mode not supported.       |

### 52.5.35 status\_t PXP\_StartPictureCopy ( PXP\_Type \* *base*, const pxp\_pic\_config\_t \* *config* )

This function copies a rectangle from one buffer to another buffer.



```
|
| |
| destPicthBytes |
+-----
```

## Note

This function resets the old PXP settings, which means the settings like rotate, flip, will be reseted to disabled status.

## Parameters

|               |                                                      |
|---------------|------------------------------------------------------|
| <i>base</i>   | PXP peripheral base address.                         |
| <i>config</i> | Pointer to the picture copy configuration structure. |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Successfully started the copy process. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                      |

**52.5.36** `status_t PXP_StartMemCopy ( PXP_Type * base, uint32_t srcAddr,  
uint32_t destAddr, uint32_t size )`

## Note

The copy size should be 512 byte aligned.

This function resets the old PXP settings, which means the settings like rotate, flip, will be reseted to disabled status.

## Parameters

|                 |                                                     |
|-----------------|-----------------------------------------------------|
| <i>base</i>     | PXP peripheral base address.                        |
| <i>srcAddr</i>  | Source memory address.                              |
| <i>destAddr</i> | Destination memory address.                         |
| <i>size</i>     | How many bytes to copy, should be 512 byte aligned. |

## Return values

|                                |                                        |
|--------------------------------|----------------------------------------|
| <i>kStatus_Success</i>         | Successfully started the copy process. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                      |

## Chapter 53

# QTMR: Quad Timer Driver

### 53.1 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\_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\_preload\_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, 0, 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)  
*Enables the selected Quad Timer interrupts.*

- void [QTMR\\_DisableInterrupts](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel, uint32\_t mask)  
*Disables the selected Quad Timer interrupts.*
- uint32\_t [QTMR\\_GetEnabledInterrupts](#) (TMR\_Type \*base, [qtmr\\_channel\\_selection\\_t](#) channel)  
*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.*
- 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.*

## 53.2 Data Structure Documentation

### 53.2.1 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.*

## 53.3 Enumeration Type Documentation

### 53.3.1 `enum qtmr_primary_count_source_t`

Enumerator

**`kQTMR_ClockCounter0InputPin`** Use counter 0 input pin.  
**`kQTMR_ClockCounter1InputPin`** Use counter 1 input pin.  
**`kQTMR_ClockCounter2InputPin`** Use counter 2 input pin.  
**`kQTMR_ClockCounter3InputPin`** Use counter 3 input pin.  
**`kQTMR_ClockCounter0Output`** Use counter 0 output.  
**`kQTMR_ClockCounter1Output`** Use counter 1 output.  
**`kQTMR_ClockCounter2Output`** Use counter 2 output.  
**`kQTMR_ClockCounter3Output`** 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.

### 53.3.2 `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.

### 53.3.3 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)

### 53.3.4 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.

### 53.3.5 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.

### 53.3.6 enum qtmr\_preload\_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.

### 53.3.7 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.

### 53.3.8 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.

### 53.3.9 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.

### 53.3.10 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.

### 53.3.11 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.

## 53.4 Function Documentation

### 53.4.1 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 |

### 53.4.2 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          |

### 53.4.3 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. |
|---------------|------------------------------------------------|

#### 53.4.4 **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.

#### 53.4.5 **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             |

#### 53.4.6 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> |

#### 53.4.7 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> |

#### 53.4.8 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](#)

### 53.4.9 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](#)

### 53.4.10 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> |

### 53.4.11 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     |

### 53.4.12 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

### 53.4.13 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            |

**53.4.14** `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          |

**53.4.15** `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> |

**53.4.16** `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> |
|-------------|---------------------------------------------------------------------------------------------------------|

## Chapter 54

# RTWDOG: 32-bit Watchdog Timer

### 54.1 Overview

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

### 54.2 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_CountToMsec` (RTWDOG\_Type \*base, uint32\_t count, uint32\_t clock-FreqInHz)  
*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)

- *Sets the RTWDOG timeout value.*  
static void [RTWDOG\\_SetWindowValue](#) (RTWDOG\_Type \*base, uint16\_t windowValue)
- *Sets the RTWDOG window 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.*

## 54.3 Data Structure Documentation

### 54.3.1 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.*

### 54.3.2 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.*

## 54.4 Macro Definition Documentation

### 54.4.1 #define FSL\_RTWD0G\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 2))

## 54.5 Enumeration Type Documentation

### 54.5.1 enum rtwdog\_clock\_source\_t

Enumerator

*kRTWD0G\_ClockSource0* Clock source 0.  
*kRTWD0G\_ClockSource1* Clock source 1.  
*kRTWD0G\_ClockSource2* Clock source 2.  
*kRTWD0G\_ClockSource3* Clock source 3.

### 54.5.2 enum rtwdog\_clock\_prescaler\_t

Enumerator

*kRTWD0G\_ClockPrescalerDivide1* Divided by 1.  
*kRTWD0G\_ClockPrescalerDivide256* Divided by 256.

### 54.5.3 enum rtwdog\_test\_mode\_t

Enumerator

*kRTWD0G\_TestModeDisabled* Test Mode disabled.  
*kRTWD0G\_UserModeEnabled* User Mode enabled.  
*kRTWD0G\_LowByteTest* Test Mode enabled, only low byte is used.  
*kRTWD0G\_HighByteTest* Test Mode enabled, only high byte is used.

### 54.5.4 enum \_rtwd0g\_interrupt\_enable\_t

This structure contains the settings for all of the RTWD0G interrupt configurations.

Enumerator

*kRTWD0G\_InterruptEnable* Interrupt is generated before forcing a reset.

### 54.5.5 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.

## 54.6 Function Documentation

### 54.6.1 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](#)

### 54.6.2 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, enable-Update 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. |

**54.6.3 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. |
|-------------|---------------------------------|

**54.6.4 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. |
|-------------|---------------------------------|

**54.6.5 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 |
|-------------|--------------------------------|

#### 54.6.6 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> |

#### 54.6.7 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> |

#### 54.6.8 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.

#### 54.6.9 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. |

#### 54.6.10 static uint32\_t RTWDOG\_CountToMsec ( 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. |

#### 54.6.11 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> |

#### 54.6.12 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. |

#### 54.6.13 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.            |

#### 54.6.14 \_\_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 |
|-------------|--------------------------------|

#### 54.6.15 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 |
|-------------|--------------------------------|

#### 54.6.16 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 55

# RDC: Resource Domain Controller

### 55.1 Overview

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

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

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

### Data Structures

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

### Enumerations

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

### Functions

- void [RDC\\_Init](#) (RDC\_Type \*base)  
*Initializes the RDC module.*
- void [RDC\\_Deinit](#) (RDC\_Type \*base)  
*De-initializes the RDC module.*
- void [RDC\\_GetHardwareConfig](#) (RDC\_Type \*base, [rdc\\_hardware\\_config\\_t](#) \*config)  
*Gets the RDC hardware configuration.*

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

## 55.2 Data Structure Documentation

### 55.2.1 struct rdc\_hardware\_config\_t

#### Data Fields

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

#### Field Documentation

- (1) uint32\_t rdc\_hardware\_config\_t::domainNumber
- (2) uint32\_t rdc\_hardware\_config\_t::masterNumber
- (3) uint32\_t rdc\_hardware\_config\_t::periphNumber
- (4) uint32\_t rdc\_hardware\_config\_t::memNumber

### 55.2.2 struct rdc\_domain\_assignment\_t

#### Data Fields

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

#### Field Documentation

- (1) uint32\_t rdc\_domain\_assignment\_t::domainId
- (2) uint32\_t rdc\_domain\_assignment\_t::\_\_pad0\_\_
- (3) uint32\_t rdc\_domain\_assignment\_t::lock

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

#### Data Fields

- rdc\_periph\_t [periph](#)  
*Peripheral name.*

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

#### Field Documentation

- (1) `rdc_periph_t rdc_periph_access_config_t::periph`
- (2) `bool rdc_periph_access_config_t::lock`
- (3) `bool rdc_periph_access_config_t::enableSema`
- (4) `uint16_t rdc_periph_access_config_t::policy`

#### 55.2.4 struct rdc\_mem\_access\_config\_t

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

#### Data Fields

- `rdc_mem_t` [mem](#)  
*Memory region descriptor name.*
- bool [lock](#)  
*Lock the configuration.*
- uint64\_t [baseAddress](#)  
*Start address of the memory region.*
- uint64\_t [endAddress](#)  
*End address of the memory region.*
- uint16\_t [policy](#)  
*Access policy.*

#### Field Documentation

- (1) `rdc_mem_t rdc_mem_access_config_t::mem`
- (2) `bool rdc_mem_access_config_t::lock`
- (3) `uint64_t rdc_mem_access_config_t::baseAddress`
- (4) `uint64_t rdc_mem_access_config_t::endAddress`
- (5) `uint16_t rdc_mem_access_config_t::policy`

## 55.2.5 struct rdc\_mem\_status\_t

### Data Fields

- bool [hasViolation](#)  
*Violating happens or not.*
- uint8\_t [domainID](#)  
*Violating Domain ID.*
- uint64\_t [address](#)  
*Violating Address.*

### Field Documentation

(1) bool rdc\_mem\_status\_t::hasViolation

(2) uint8\_t rdc\_mem\_status\_t::domainID

(3) uint64\_t rdc\_mem\_status\_t::address

## 55.3 Enumeration Type Documentation

### 55.3.1 enum \_rdc\_interrupts

#### Enumerator

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

### 55.3.2 enum \_rdc\_flags

#### Enumerator

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

### 55.3.3 enum \_rdc\_access\_policy

#### Enumerator

***kRDC\_NoAccess*** Could not read or write.  
***kRDC\_WriteOnly*** Write only.  
***kRDC\_ReadOnly*** Read only.  
***kRDC\_ReadWrite*** Read and write.

## 55.4 Function Documentation



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

This function enables the RDC clock.

## Parameters

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

**55.4.2 void RDC\_Deinit ( RDC\_Type \* *base* )**

This function disables the RDC clock.

## Parameters

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

**55.4.3 void RDC\_GetHardwareConfig ( RDC\_Type \* *base*, rdc\_hardware\_config\_t \* *config* )**

This function gets the RDC hardware configurations, including number of bus masters, number of domains, number of memory regions and number of peripherals.

## Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | RDC peripheral base address.                       |
| <i>config</i> | Pointer to the structure to get the configuration. |

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

## Parameters

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>base</i> | RDC peripheral base address.                                                      |
| <i>mask</i> | Interrupts to enable, it is OR'ed value of enum <a href="#">_rdc_interrupts</a> . |

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

## Parameters

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| <i>base</i> | RDC peripheral base address.                                                       |
| <i>mask</i> | Interrupts to disable, it is OR'ed value of enum <a href="#">_rdc_interrupts</a> . |

#### 55.4.6 static uint32\_t RDC\_GetInterruptStatus ( RDC\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

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

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

## Parameters

|             |                                                                              |
|-------------|------------------------------------------------------------------------------|
| <i>base</i> | RDC peripheral base address.                                                 |
| <i>mask</i> | Status to clear, it is OR'ed value of enum <a href="#">_rdc_interrupts</a> . |

#### 55.4.8 static uint32\_t RDC\_GetStatus ( RDC\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

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

#### 55.4.9 static void RDC\_ClearStatus ( RDC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                             |
|-------------|-----------------------------------------------------------------------------|
| <i>base</i> | RDC peripheral base address.                                                |
| <i>mask</i> | RDC status to clear, it is OR'ed value of enum <a href="#">_rdc_flags</a> . |

**55.4.10 void RDC\_SetMasterDomainAssignment ( RDC\_Type \* *base*, rdc\_master\_t *master*, const rdc\_domain\_assignment\_t \* *domainAssignment* )**

## Parameters

|                          |                              |
|--------------------------|------------------------------|
| <i>base</i>              | RDC peripheral base address. |
| <i>master</i>            | Which master to set.         |
| <i>domain-Assignment</i> | Pointer to the assignment.   |

**55.4.11 void RDC\_GetDefaultMasterDomainAssignment ( rdc\_domain\_assignment\_t \* *domainAssignment* )**

The default configuration is:

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

## Parameters

|                          |                            |
|--------------------------|----------------------------|
| <i>domain-Assignment</i> | Pointer to the assignment. |
|--------------------------|----------------------------|

**55.4.12 static void RDC\_LockMasterDomainAssignment ( RDC\_Type \* *base*, rdc\_master\_t *master* ) [inline], [static]**

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

## Parameters

---

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

#### 55.4.13 void RDC\_SetPeriphAccessConfig ( RDC\_Type \* *base*, const rdc\_periph\_access\_config\_t \* *config* )

Parameters

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

#### 55.4.14 void RDC\_GetDefaultPeriphAccessConfig ( rdc\_periph\_access\_config\_t \* *config* )

The default configuration is:

```
config->lock = false;
config->enableSema = false;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

Parameters

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

#### 55.4.15 static void RDC\_LockPeriphAccessConfig ( RDC\_Type \* *base*, rdc\_periph\_t *periph* ) [inline], [static]

Once locked, it could not be unlocked until reset.

Parameters

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

|               |                           |
|---------------|---------------------------|
| <i>periph</i> | Which peripheral to lock. |
|---------------|---------------------------|

**55.4.16** `static uint8_t RDC_GetPeriphAccessPolicy ( RDC_Type * base,  
rdc_periph_t periph, uint8_t domainId ) [inline], [static]`

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | RDC peripheral base address. |
| <i>periph</i>   | Which peripheral to get.     |
| <i>domainId</i> | Get policy for which domain. |

Returns

Access policy, see [\\_rdc\\_access\\_policy](#).

**55.4.17** `void RDC_SetMemAccessConfig ( RDC_Type * base, const  
rdc_mem_access_config_t * config )`

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

Parameters

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

**55.4.18** `void RDC_GetDefaultMemAccessConfig ( rdc_mem_access_config_t *  
config )`

The default configuration is:

```
config->lock = false;
config->baseAddress = 0;
config->endAddress = 0;
config->policy = RDC_ACCESS_POLICY(0, kRDC_ReadWrite) |
 RDC_ACCESS_POLICY(1, kRDC_ReadWrite) |
 RDC_ACCESS_POLICY(2, kRDC_ReadWrite) |
 RDC_ACCESS_POLICY(3, kRDC_ReadWrite);
```

## Parameters

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

#### 55.4.19 static void RDC\_LockMemAccessConfig ( RDC\_Type \* *base*, rdc\_mem\_t *mem* ) [inline], [static]

Once locked, it could not be unlocked until reset. After locked, you can only call [RDC\\_SetMemAccessValid](#) to enable the configuration, but can not disable it or change other settings.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | RDC peripheral base address. |
| <i>mem</i>  | Which memory region to lock. |

#### 55.4.20 static void RDC\_SetMemAccessValid ( RDC\_Type \* *base*, rdc\_mem\_t *mem*, bool *valid* ) [inline], [static]

## Parameters

|              |                                          |
|--------------|------------------------------------------|
| <i>base</i>  | RDC peripheral base address.             |
| <i>mem</i>   | Which memory region to operate.          |
| <i>valid</i> | Pass in true to valid, false to invalid. |

#### 55.4.21 void RDC\_GetMemViolationStatus ( RDC\_Type \* *base*, rdc\_mem\_t *mem*, rdc\_mem\_status\_t \* *status* )

The first access violation is captured. Subsequent violations are ignored until the status register is cleared. Contents are cleared upon reading the register. Clearing of contents occurs only when the status is read by the memory region's associated domain ID(s).

## Parameters

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

|               |                             |
|---------------|-----------------------------|
| <i>mem</i>    | Which memory region to get. |
| <i>status</i> | The returned status.        |

**55.4.22 static void RDC\_ClearMemViolationFlag ( RDC\_Type \* *base*, rdc\_mem\_t *mem* ) [inline], [static]**

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | RDC peripheral base address.  |
| <i>mem</i>  | Which memory region to clear. |

**55.4.23 static uint8\_t RDC\_GetMemAccessPolicy ( RDC\_Type \* *base*, rdc\_mem\_t *mem*, uint8\_t *domainId* ) [inline], [static]**

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | RDC peripheral base address. |
| <i>mem</i>      | Which memory region to get.  |
| <i>domainId</i> | Get policy for which domain. |

Returns

Access policy, see [\\_rdc\\_access\\_policy](#).

**55.4.24 static uint8\_t RDC\_GetCurrentMasterDomainId ( RDC\_Type \* *base* ) [inline], [static]**

This function returns the domain ID of the current bus master.

Parameters

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

Returns

Domain ID of current bus master.



## Chapter 56

# RDC\_SEMA42: Hardware Semaphores Driver

### 56.1 Overview

The MCUXpresso SDK provides a driver for the RDC\_SEMA42 module of MCUXpresso SDK devices.

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

Before using the RDC\_SEMA42, call the [RDC\\_SEMA42\\_Init\(\)](#) function to initialize the module. Note that this function only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either the [RDC\\_SEMA42\\_ResetGate\(\)](#) or [RDC\\_SEMA42\\_ResetAllGates\(\)](#) functions. The function [RDC\\_SEMA42\\_Deinit\(\)](#) deinitializes the RDC\_SEMA42.

The RDC\_SEMA42 provides two functions to lock the RDC\_SEMA42 gate. The function [RDC\\_SEMA42\\_TryLock\(\)](#) tries to lock the gate. If the gate has been locked by another processor, this function returns an error immediately. The function [RDC\\_SEMA42\\_Lock\(\)](#) is a blocking method, which waits until the gate is free and locks it.

The [RDC\\_SEMA42\\_Unlock\(\)](#) unlocks the RDC\_SEMA42 gate. The gate can only be unlocked by the processor which locked it. If the gate is not locked by the current processor, this function takes no effect. The function [RDC\\_SEMA42\\_GetGateStatus\(\)](#) returns a status whether the gate is unlocked and which processor locks the gate. The function [RDC\\_SEMA42\\_GetLockDomainID\(\)](#) returns the ID of the domain which has locked the gate.

The RDC\_SEMA42 gate can be reset to unlock forcefully. The function [RDC\\_SEMA42\\_ResetGate\(\)](#) resets a specific gate. The function [RDC\\_SEMA42\\_ResetAllGates\(\)](#) resets all gates.

### Macros

- #define [RDC\\_SEMA42\\_GATE\\_NUM\\_RESET\\_ALL](#) (64U)  
*The number to reset all RDC\_SEMA42 gates.*
- #define [RDC\\_SEMA42\\_GATEn](#)(base, n) (((volatile uint8\_t \*)&((base)->GATE0)))[(n)]  
*RDC\_SEMA42 gate n register address.*
- #define [RDC\\_SEMA42\\_GATE\\_COUNT](#) (64U)  
*RDC\_SEMA42 gate count.*

### Functions

- void [RDC\\_SEMA42\\_Init](#) (RDC\_SEMAPHORE\_Type \*base)  
*Initializes the RDC\_SEMA42 module.*
- void [RDC\\_SEMA42\\_Deinit](#) (RDC\_SEMAPHORE\_Type \*base)  
*De-initializes the RDC\_SEMA42 module.*
- [status\\_t](#) [RDC\\_SEMA42\\_TryLock](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum, uint8\_t masterIndex, uint8\_t domainId)  
*Tries to lock the RDC\_SEMA42 gate.*

- void [RDC\\_SEMA42\\_Lock](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum, uint8\_t masterIndex, uint8\_t domainId)  
*Locks the RDC\_SEMA42 gate.*
- static void [RDC\\_SEMA42\\_Unlock](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Unlocks the RDC\_SEMA42 gate.*
- static int32\_t [RDC\\_SEMA42\\_GetLockMasterIndex](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Gets which master has currently locked the gate.*
- int32\_t [RDC\\_SEMA42\\_GetLockDomainID](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Gets which domain has currently locked the gate.*
- status\_t [RDC\\_SEMA42\\_ResetGate](#) (RDC\_SEMAPHORE\_Type \*base, uint8\_t gateNum)  
*Resets the RDC\_SEMA42 gate to an unlocked status.*
- static status\_t [RDC\\_SEMA42\\_ResetAllGates](#) (RDC\_SEMAPHORE\_Type \*base)  
*Resets all RDC\_SEMA42 gates to an unlocked status.*

## Driver version

- #define [FSL\\_RDC\\_SEMA42\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 3))  
*RDC\_SEMA42 driver version.*

## 56.2 Macro Definition Documentation

### 56.2.1 #define RDC\_SEMA42\_GATE\_NUM\_RESET\_ALL (64U)

### 56.2.2 #define RDC\_SEMA42\_GATEn( base, n ) (((volatile uint8\_t \*)(&((base)->GATE0)))[(n)])

### 56.2.3 #define RDC\_SEMA42\_GATE\_COUNT (64U)

## 56.3 Function Documentation

### 56.3.1 void RDC\_SEMA42\_Init ( RDC\_SEMAPHORE\_Type \* base )

This function initializes the RDC\_SEMA42 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either RDC\_SEMA42\_ResetGate or RDC\_SEMA42\_ResetAllGates function.

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | RDC_SEMA42 peripheral base address. |
|-------------|-------------------------------------|

### 56.3.2 void RDC\_SEMA42\_Deinit ( RDC\_SEMAPHORE\_Type \* base )

This function de-initializes the RDC\_SEMA42 module. It only disables the clock.

## Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | RDC_SEMA42 peripheral base address. |
|-------------|-------------------------------------|

### 56.3.3 **status\_t RDC\_SEMA42\_TryLock ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *masterIndex*, uint8\_t *domainId* )**

This function tries to lock the specific RDC\_SEMA42 gate. If the gate has been locked by another processor, this function returns an error code.

## Parameters

|                    |                                     |
|--------------------|-------------------------------------|
| <i>base</i>        | RDC_SEMA42 peripheral base address. |
| <i>gateNum</i>     | Gate number to lock.                |
| <i>masterIndex</i> | Current processor master index.     |
| <i>domainId</i>    | Current processor domain ID.        |

## Return values

|                        |                                                   |
|------------------------|---------------------------------------------------|
| <i>kStatus_Success</i> | Lock the sema42 gate successfully.                |
| <i>kStatus_Failed</i>  | Sema42 gate has been locked by another processor. |

### 56.3.4 **void RDC\_SEMA42\_Lock ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *masterIndex*, uint8\_t *domainId* )**

This function locks the specific RDC\_SEMA42 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

## Parameters

|                    |                                     |
|--------------------|-------------------------------------|
| <i>base</i>        | RDC_SEMA42 peripheral base address. |
| <i>gateNum</i>     | Gate number to lock.                |
| <i>masterIndex</i> | Current processor master index.     |
| <i>domainId</i>    | Current processor domain ID.        |

### 56.3.5 static void RDC\_SEMA42\_Unlock ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function unlocks the specific RDC\_SEMA42 gate. It only writes unlock value to the RDC\_SEMA42 gate register. However, it does not check whether the RDC\_SEMA42 gate is locked by the current processor or not. As a result, if the RDC\_SEMA42 gate is not locked by the current processor, this function has no effect.

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | RDC_SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number to unlock.              |

### 56.3.6 static int32\_t RDC\_SEMA42\_GetLockMasterIndex ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | RDC_SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number.                        |

Returns

Return -1 if the gate is not locked by any master, otherwise return the master index.

### 56.3.7 int32\_t RDC\_SEMA42\_GetLockDomainID ( RDC\_SEMAPHORE\_Type \* *base*, uint8\_t *gateNum* )

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | RDC_SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number.                        |

Returns

Return -1 if the gate is not locked by any domain, otherwise return the domain ID.

### 56.3.8 `status_t RDC_SEMA42_ResetGate ( RDC_SEMAPHORE_Type * base, uint8_t gateNum )`

This function resets a RDC\_SEMA42 gate to an unlocked status.

## Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | RDC_SEMA42 peripheral base address. |
| <i>gateNum</i> | Gate number.                        |

## Return values

|                        |                                        |
|------------------------|----------------------------------------|
| <i>kStatus_Success</i> | RDC_SEMA42 gate is reset successfully. |
| <i>kStatus_Failed</i>  | Some other reset process is ongoing.   |

### 56.3.9 static status\_t RDC\_SEMA42\_ResetAllGates ( RDC\_SEMAPHORE\_Type \* *base* ) [inline], [static]

This function resets all RDC\_SEMA42 gate to an unlocked status.

## Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | RDC_SEMA42 peripheral base address. |
|-------------|-------------------------------------|

## Return values

|                                     |                                      |
|-------------------------------------|--------------------------------------|
| <i>kStatus_Success</i>              | RDC_SEMA42 is reset successfully.    |
| <i>kStatus_RDC_SEMA42_-Reseting</i> | Some other reset process is ongoing. |

## Chapter 57

# SAI: Serial Audio Interface

### 57.1 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.

### 57.2 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`

## 57.3 Typical use case

### 57.3.1 SAI Send/receive using an interrupt method

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

### 57.3.2 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](#)



## 57.4 SAI Driver

### 57.4.1 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_FifoCombineModeEnabledOnReadWrite` }

*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, 3, 4)`)  
*Version 2.3.4.*

## Initialization and deinitialization

- void [SAI\\_TxInit](#) (I2S\_Type \*base, const [sai\\_config\\_t](#) \*config)  
*Initializes the SAI Tx peripheral.*
- void [SAI\\_RxInit](#) (I2S\_Type \*base, const [sai\\_config\\_t](#) \*config)  
*Initializes the SAI Rx peripheral.*
- void [SAI\\_TxGetDefaultConfig](#) ([sai\\_config\\_t](#) \*config)  
*Sets the SAI Tx configuration structure to default values.*
- void [SAI\\_RxGetDefaultConfig](#) ([sai\\_config\\_t](#) \*config)  
*Sets the SAI Rx configuration structure to default values.*
- void [SAI\\_Init](#) (I2S\_Type \*base)  
*Initializes the SAI peripheral.*
- void [SAI\\_Deinit](#) (I2S\_Type \*base)  
*De-initializes the SAI peripheral.*
- void [SAI\\_TxReset](#) (I2S\_Type \*base)  
*Resets the SAI Tx.*
- void [SAI\\_RxReset](#) (I2S\_Type \*base)  
*Resets the SAI Rx.*
- void [SAI\\_TxEnable](#) (I2S\_Type \*base, bool enable)  
*Enables/disables the SAI Tx.*
- void [SAI\\_RxEnable](#) (I2S\_Type \*base, bool enable)  
*Enables/disables the SAI Rx.*
- static void [SAI\\_TxSetBitClockDirection](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave)  
*Set Rx bit clock direction.*
- static void [SAI\\_RxSetBitClockDirection](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave)  
*Set Rx bit clock direction.*
- static void [SAI\\_RxSetFrameSyncDirection](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave)  
*Set Rx frame sync direction.*
- static void [SAI\\_TxSetFrameSyncDirection](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave)  
*Set Tx frame sync direction.*
- void [SAI\\_TxSetBitClockRate](#) (I2S\_Type \*base, uint32\_t sourceClockHz, uint32\_t sampleRate, uint32\_t bitWidth, uint32\_t channelNumbers)  
*Transmitter bit clock rate configurations.*
- void [SAI\\_RxSetBitClockRate](#) (I2S\_Type \*base, uint32\_t sourceClockHz, uint32\_t sampleRate, uint32\_t bitWidth, uint32\_t channelNumbers)  
*Receiver bit clock rate configurations.*
- void [SAI\\_TxSetBitclockConfig](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave, [sai\\_bit\\_clock\\_t](#) \*config)  
*Transmitter Bit clock configurations.*
- void [SAI\\_RxSetBitclockConfig](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) masterSlave, [sai\\_bit\\_clock\\_t](#) \*config)  
*Receiver Bit clock configurations.*
- void [SAI\\_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.*

`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` type)

*Do software reset or FIFO reset.*

- void `SAI_RxSoftwareReset` (I2S\_Type \*base, `sai_reset_type_t` type)

*Do software reset or FIFO reset.*

- void `SAI_TxSetChannelFIFOMask` (I2S\_Type \*base, `uint8_t` mask)

*Set the Tx channel FIFO enable mask.*

- void `SAI_RxSetChannelFIFOMask` (I2S\_Type \*base, `uint8_t` mask)

*Set the Rx channel FIFO enable mask.*

- void `SAI_TxSetDataOrder` (I2S\_Type \*base, `sai_data_order_t` order)

*Set the Tx data order.*

- void `SAI_RxSetDataOrder` (I2S\_Type \*base, `sai_data_order_t` order)

*Set the Rx data order.*

- void `SAI_TxSetBitClockPolarity` (I2S\_Type \*base, `sai_clock_polarity_t` polarity)

- *Set the Tx data order.*  
void [SAI\\_RxSetBitClockPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)
- *Set the Rx data order.*  
void [SAI\\_TxSetFrameSyncPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)
- *Set the Tx data order.*  
void [SAI\\_RxSetFrameSyncPolarity](#) (I2S\_Type \*base, [sai\\_clock\\_polarity\\_t](#) polarity)
- *Set the Rx data order.*  
void [SAI\\_TxSetFIFOPacking](#) (I2S\_Type \*base, [sai\\_fifo\\_packing\\_t](#) pack)
- *Set Tx FIFO packing feature.*  
void [SAI\\_RxSetFIFOPacking](#) (I2S\_Type \*base, [sai\\_fifo\\_packing\\_t](#) pack)
- *Set Rx FIFO packing feature.*  
static void [SAI\\_TxSetFIFOErrorContinue](#) (I2S\_Type \*base, bool isEnabled)
- *Set Tx FIFO error continue.*  
static void [SAI\\_RxSetFIFOErrorContinue](#) (I2S\_Type \*base, bool isEnabled)
- *Set Rx FIFO error continue.*

## Interrupts

- static void [SAI\\_TxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables the SAI Tx interrupt requests.*
- static void [SAI\\_RxEnableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Enables the SAI Rx interrupt requests.*
- static void [SAI\\_TxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables the SAI Tx interrupt requests.*
- static void [SAI\\_RxDisableInterrupts](#) (I2S\_Type \*base, uint32\_t mask)  
*Disables the SAI Rx interrupt requests.*

## DMA Control

- static void [SAI\\_TxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SAI Tx DMA requests.*
- static void [SAI\\_RxEnableDMA](#) (I2S\_Type \*base, uint32\_t mask, bool enable)  
*Enables/disables the SAI Rx DMA requests.*
- static uint32\_t [SAI\\_TxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Tx data register address.*
- static uint32\_t [SAI\\_RxGetDataRegisterAddress](#) (I2S\_Type \*base, uint32\_t channel)  
*Gets the SAI Rx data register address.*

## Bus Operations

- void [SAI\\_TxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_RxSetFormat](#) (I2S\_Type \*base, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*



- void [SAI\\_WriteBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- void [SAI\\_WriteMultiChannelBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t channelMask, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Sends data to multi channel using a blocking method.*
- static void [SAI\\_WriteData](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t data)  
*Writes data into SAI FIFO.*
- void [SAI\\_ReadBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Receives data using a blocking method.*
- void [SAI\\_ReadMultiChannelBlocking](#) (I2S\_Type \*base, uint32\_t channel, uint32\_t channelMask, uint32\_t bitWidth, uint8\_t \*buffer, uint32\_t size)  
*Receives multi channel data using a blocking method.*
- static uint32\_t [SAI\\_ReadData](#) (I2S\_Type \*base, uint32\_t channel)  
*Reads data from the SAI FIFO.*

## Transactional

- void [SAI\\_TransferTxCreateHandle](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_callback\_t callback, void \*userData)  
*Initializes the SAI Tx handle.*
- void [SAI\\_TransferRxCreateHandle](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_callback\_t callback, void \*userData)  
*Initializes the SAI Rx handle.*
- void [SAI\\_TransferTxSetConfig](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transceiver\_t \*config)  
*SAI transmitter transfer configurations.*
- void [SAI\\_TransferRxSetConfig](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transceiver\_t \*config)  
*SAI receiver transfer configurations.*
- [status\\_t](#) [SAI\\_TransferTxSetFormat](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- [status\\_t](#) [SAI\\_TransferRxSetFormat](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_format\_t \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- [status\\_t](#) [SAI\\_TransferSendNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking send transfer on SAI.*
- [status\\_t](#) [SAI\\_TransferReceiveNonBlocking](#) (I2S\_Type \*base, sai\_handle\_t \*handle, sai\_transfer\_t \*xfer)  
*Performs an interrupt non-blocking receive transfer on SAI.*
- [status\\_t](#) [SAI\\_TransferGetSendCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a set byte count.*
- [status\\_t](#) [SAI\\_TransferGetReceiveCount](#) (I2S\_Type \*base, sai\_handle\_t \*handle, size\_t \*count)  
*Gets a received byte count.*
- void [SAI\\_TransferAbortSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Aborts the current send.*
- void [SAI\\_TransferAbortReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)



- *Aborts the current IRQ receive.*
- void [SAI\\_TransferTerminateSend](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Terminate all SAI send.*
- void [SAI\\_TransferTerminateReceive](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Terminate all SAI receive.*
- void [SAI\\_TransferTxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Tx interrupt handler.*
- void [SAI\\_TransferRxHandleIRQ](#) (I2S\_Type \*base, sai\_handle\_t \*handle)  
*Rx interrupt handler.*

## 57.4.2 Data Structure Documentation

### 57.4.2.1 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.*

### 57.4.2.2 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`

#### 57.4.2.3 struct sai\_fifo\_t

### Data Fields

- `bool fifoContinueOnError`  
*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*

#### 57.4.2.4 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`

#### 57.4.2.5 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.*
- `sai_clock_polarity_t frameSyncPolarity`  
*frame sync polarity*

### 57.4.2.6 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*

### 57.4.2.7 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.*

### 57.4.2.8 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

### 57.4.2.9 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, refernece \_sai\_channel\_mask*
- uint8\_t [endChannel](#)  
*end channel number*
- uint8\_t [channelNums](#)  
*Total enabled channel numbers.*
- [sai\\_transfer\\_t](#) [saiQueue](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [[SAI\\_XFER\\_QUEUE\\_SIZE](#)]  
*Data bytes need to transfer.*
- volatile uint8\_t [queueUser](#)  
*Index for user to queue transfer.*
- volatile uint8\_t [queueDriver](#)  
*Index for driver to get the transfer data and size.*
- uint8\_t [watermark](#)  
*Watermark value.*

### 57.4.3 Macro Definition Documentation

### 57.4.3.1 #define SAI\_XFER\_QUEUE\_SIZE (4U)

## 57.4.4 Enumeration Type Documentation

### 57.4.4.1 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.

### 57.4.4.2 anonymous enum

Enumerator

*kSAI\_Channel0Mask* channel 0 mask value  
*kSAI\_Channel1Mask* channel 1 mask value  
*kSAI\_Channel2Mask* channel 2 mask value  
*kSAI\_Channel3Mask* channel 3 mask value  
*kSAI\_Channel4Mask* channel 4 mask value  
*kSAI\_Channel5Mask* channel 5 mask value  
*kSAI\_Channel6Mask* channel 6 mask value  
*kSAI\_Channel7Mask* channel 7 mask value

### 57.4.4.3 enum sai\_protocol\_t

Enumerator

*kSAI\_BusLeftJustified* Uses left justified format.  
*kSAI\_BusRightJustified* Uses right justified format.  
*kSAI\_BusI2S* Uses I2S format.  
*kSAI\_BusPCMA* Uses I2S PCM A format.  
*kSAI\_BusPCMB* Uses I2S PCM B format.

### 57.4.4.4 enum sai\_master\_slave\_t

Enumerator

*kSAI\_Master* Master mode include bclk and frame sync.

*kSAI\_Slave* Slave mode include bclk and frame sync.

*kSAI\_Bclk\_Master\_FrameSync\_Slave* bclk in master mode, frame sync in slave mode

*kSAI\_Bclk\_Slave\_FrameSync\_Master* bclk in slave mode, frame sync in master mode

#### 57.4.4.5 enum sai\_mono\_stereo\_t

Enumerator

*kSAI\_Stereo* Stereo sound.

*kSAI\_MonoRight* Only Right channel have sound.

*kSAI\_MonoLeft* Only left channel have sound.

#### 57.4.4.6 enum sai\_data\_order\_t

Enumerator

*kSAI\_DataLSB* LSB bit transferred first.

*kSAI\_DataMSB* MSB bit transferred first.

#### 57.4.4.7 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.

#### 57.4.4.8 enum sai\_sync\_mode\_t

Enumerator

*kSAI\_ModeAsync* Asynchronous mode.

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

#### 57.4.4.9 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.

#### 57.4.4.10 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.

#### 57.4.4.11 anonymous enum

Enumerator

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

#### 57.4.4.12 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.

#### 57.4.4.13 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.

**57.4.4.14 enum sai\_fifo\_packing\_t**

Enumerator

***kSAI\_FifoPackingDisabled*** Packing disabled.  
***kSAI\_FifoPacking8bit*** 8 bit packing enabled  
***kSAI\_FifoPacking16bit*** 16bit packing enabled

**57.4.4.15 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.

**57.4.4.16 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.

**57.4.4.17 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



**57.4.4.18 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

**57.4.4.19 enum sai\_transceiver\_type\_t**

Enumerator

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

**57.4.4.20 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.

**57.4.5 Function Documentation****57.4.5.1 void SAI\_TxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )**

**Deprecated** Do not use this function. It has been superceded by [SAI\\_Init](#)

Un-gates the SAI clock, resets the module, and configures SAI Tx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_TxGetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAIM module can cause a hard fault because the clock is not enabled.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

**57.4.5.2 void SAI\_RxInit ( I2S\_Type \* *base*, const sai\_config\_t \* *config* )**

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

Ungates the SAI clock, resets the module, and configures the SAI Rx with a configuration structure. The configuration structure can be custom filled or set with default values by [SAI\\_RxGetDefaultConfig\(\)](#).

## Note

This API should be called at the beginning of the application to use the SAI driver. Otherwise, accessing the SAI module can cause a hard fault because the clock is not enabled.

## Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SAI base pointer             |
| <i>config</i> | SAI configuration structure. |

**57.4.5.3 void SAI\_TxGetDefaultConfig ( sai\_config\_t \* *config* )**

**Deprecated** Do not use this function. It has been superseded by [SAI\\_GetClassicI2SConfig](#), [SAI\\_GetLeftJustifiedConfig](#), [SAI\\_GetRightJustifiedConfig](#), [SAI\\_GetDSPConfig](#), [SAI\\_GetTDMConfig](#)

This API initializes the configuration structure for use in [SAI\\_TxConfig\(\)](#). The initialized structure can remain unchanged in [SAI\\_TxConfig\(\)](#), or it can be modified before calling [SAI\\_TxConfig\(\)](#). This is an example.

```
sai_config_t config;
SAI_TxGetDefaultConfig(&config);
```

## Parameters

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

#### 57.4.5.4 void SAI\_RxGetDefaultConfig ( sai\_config\_t \* *config* )

**Deprecated** Do not use this function. It has been superseded by [SAI\\_GetClassicI2SConfig](#), [SAI\\_GetLeftJustifiedConfig](#) , [SAI\\_GetRightJustifiedConfig](#), [SAI\\_GetDSPConfig](#), [SAI\\_GetTDMConfig](#)

This API initializes the configuration structure for use in SAI\_RxConfig(). The initialized structure can remain unchanged in SAI\_RxConfig() or it can be modified before calling SAI\_RxConfig(). This is an example.

```
sai_config_t config;
SAI_RxGetDefaultConfig(&config);
```

Parameters

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

#### 57.4.5.5 void SAI\_Init ( I2S\_Type \* *base* )

This API gates the SAI clock. The SAI module can't operate unless SAI\_Init is called to enable the clock.

Parameters

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

#### 57.4.5.6 void SAI\_Deinit ( I2S\_Type \* *base* )

This API gates the SAI clock. The SAI module can't operate unless SAI\_TxInit or SAI\_RxInit is called to enable the clock.

Parameters

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

#### 57.4.5.7 void SAI\_TxReset ( I2S\_Type \* *base* )

This function enables the software reset and FIFO reset of SAI Tx. After reset, clear the reset bit.

Parameters

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

#### 57.4.5.8 void SAI\_RxReset ( I2S\_Type \* *base* )

This function enables the software reset and FIFO reset of SAI Rx. After reset, clear the reset bit.

Parameters

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

#### 57.4.5.9 void SAI\_TxEnable ( I2S\_Type \* *base*, bool *enable* )

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SAI base pointer.                              |
| <i>enable</i> | True means enable SAI Tx, false means disable. |

#### 57.4.5.10 void SAI\_RxEnable ( I2S\_Type \* *base*, bool *enable* )

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SAI base pointer.                              |
| <i>enable</i> | True means enable SAI Rx, false means disable. |

#### 57.4.5.11 static void SAI\_TxSetBitClockDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]

Select bit clock direction, master or slave.

Parameters

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

|                    |                               |
|--------------------|-------------------------------|
| <i>masterSlave</i> | reference sai_master_slave_t. |
|--------------------|-------------------------------|

**57.4.5.12 static void SAI\_RxSetBitClockDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]**

Select bit clock direction, master or slave.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | SAI base pointer.             |
| <i>masterSlave</i> | reference sai_master_slave_t. |

**57.4.5.13 static void SAI\_RxSetFrameSyncDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]**

Select frame sync direction, master or slave.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | SAI base pointer.             |
| <i>masterSlave</i> | reference sai_master_slave_t. |

**57.4.5.14 static void SAI\_TxSetFrameSyncDirection ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave* ) [inline], [static]**

Select frame sync direction, master or slave.

Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | SAI base pointer.             |
| <i>masterSlave</i> | reference sai_master_slave_t. |

**57.4.5.15 void SAI\_TxSetBitClockRate ( I2S\_Type \* *base*, uint32\_t *sourceClockHz*, uint32\_t *sampleRate*, uint32\_t *bitWidth*, uint32\_t *channelNumbers* )**

## Parameters

|                        |                             |
|------------------------|-----------------------------|
| <i>base</i>            | SAI base pointer.           |
| <i>sourceClockHz</i>   | Bit clock source frequency. |
| <i>sampleRate</i>      | Audio data sample rate.     |
| <i>bitWidth</i>        | Audio data bitWidth.        |
| <i>channel-Numbers</i> | Audio channel numbers.      |

**57.4.5.16 void SAI\_RxSetBitClockRate ( I2S\_Type \* *base*, uint32\_t *sourceClockHz*, uint32\_t *sampleRate*, uint32\_t *bitWidth*, uint32\_t *channelNumbers* )**

## Parameters

|                        |                             |
|------------------------|-----------------------------|
| <i>base</i>            | SAI base pointer.           |
| <i>sourceClockHz</i>   | Bit clock source frequency. |
| <i>sampleRate</i>      | Audio data sample rate.     |
| <i>bitWidth</i>        | Audio data bitWidth.        |
| <i>channel-Numbers</i> | Audio channel numbers.      |

**57.4.5.17 void SAI\_TxSetBitclockConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_bit\_clock\_t \* *config* )**

## Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                          |
| <i>masterSlave</i> | master or slave.                                           |
| <i>config</i>      | bit clock other configurations, can be NULL in slave mode. |

**57.4.5.18 void SAI\_RxSetBitclockConfig ( I2S\_Type \* *base*, sai\_master\_slave\_t *masterSlave*, sai\_bit\_clock\_t \* *config* )**

## Parameters

|                    |                                                            |
|--------------------|------------------------------------------------------------|
| <i>base</i>        | SAI base pointer.                                          |
| <i>masterSlave</i> | master or slave.                                           |
| <i>config</i>      | bit clock other configurations, can be NULL in slave mode. |

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

## Parameters

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

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

## Parameters

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

**57.4.5.21 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. |

**57.4.5.22 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. |

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

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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



|               |                          |
|---------------|--------------------------|
| <i>config</i> | receiver configurations. |
|---------------|--------------------------|

**57.4.5.27 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. |

**57.4.5.28 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. |

**57.4.5.29 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. |

**57.4.5.30 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. |

**57.4.5.31 void SAI\_GetDSPConfig ( sai\_transceiver\_t \* *config*, sai\_frame\_sync\_len\_t *frameSyncWidth*, sai\_word\_width\_t *bitWidth*, sai\_mono\_stereo\_t *mode*, uint32\_t *saiChannelMask* )**

Note

DSP mode is also called PCM mode which support MODE A and MODE B, DSP/PCM MODE A configuration flow. RX is similiar but uses SAI\_RxSetConfig instead of SAI\_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
 kSAI_Stereo, channelMask)
* config->frameSync.frameSyncEarly = true;
* SAI_TxSetConfig(base, config)
*
```

DSP/PCM MODE B configuration flow for TX. RX is similiar but uses SAI\_RxSetConfig instead of SAI\_TxSetConfig:

```
* SAI_GetDSPConfig(config, kSAI_FrameSyncLenOneBitClk, bitWidth,
 kSAI_Stereo, channelMask)
* SAI_TxSetConfig(base, config)
*
```

## Parameters

|                        |                                      |
|------------------------|--------------------------------------|
| <i>config</i>          | transceiver configurations.          |
| <i>frameSync-Width</i> | length of frame sync.                |
| <i>bitWidth</i>        | audio data bitWidth.                 |
| <i>mode</i>            | audio data channel.                  |
| <i>saiChannel-Mask</i> | mask value of the channel to enable. |

**57.4.5.32** `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.

**57.4.5.33** `static void SAI_TxClearStatusFlags ( I2S_Type * base, uint32_t mask ) [inline], [static]`

## Parameters

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

**57.4.5.34** `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.

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

## Parameters

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

#### 57.4.5.36 void SAI\_TxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *type* )

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Tx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like TCR1~TCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

## Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | SAI base pointer                         |
| <i>type</i> | Reset type, FIFO reset or software reset |

#### 57.4.5.37 void SAI\_RxSoftwareReset ( I2S\_Type \* *base*, sai\_reset\_type\_t *type* )

FIFO reset means clear all the data in the FIFO, and make the FIFO pointer both to 0. Software reset means clear the Rx internal logic, including the bit clock, frame count etc. But software reset will not clear any configuration registers like RCR1~RCR5. This function will also clear all the error flags such as FIFO error, sync error etc.

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | SAI base pointer                         |
| <i>type</i> | Reset type, FIFO reset or software reset |

#### 57.4.5.38 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. |

#### 57.4.5.39 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. |

#### 57.4.5.40 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 |

#### 57.4.5.41 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 |
|--------------|-----------------------|

**57.4.5.42 void SAI\_TxSetBitClockPolarity ( I2S\_Type \* *base*, sai\_clock\_polarity\_t *polarity* )**

Parameters

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

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

Parameters

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

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

Parameters

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

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

Parameters

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

**57.4.5.46 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. |

**57.4.5.47 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. |

**57.4.5.48 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. |

**57.4.5.49 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. |

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

## Parameters

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

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

## Parameters

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

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

## Parameters

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



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

Parameters

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

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

Parameters

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

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

Parameters

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

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

#### 57.4.5.56 static uint32\_t SAI\_TxGetDataRegisterAddress ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

Returns

data register address.

#### 57.4.5.57 static uint32\_t SAI\_RxGetDataRegisterAddress ( I2S\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

This API is used to provide a transfer address for the SAI DMA transfer configuration.

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | SAI base pointer.        |
| <i>channel</i> | Which data channel used. |

Returns

data register address.

#### 57.4.5.58 void SAI\_TxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )

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

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

|                           |                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                           |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                             |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                    |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz. |

**57.4.5.59 void SAI\_RxSetFormat ( I2S\_Type \* *base*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

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

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

|                           |                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                           |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                             |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                    |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If the bit clock source is a master clock, this value should equal the masterClockHz. |

**57.4.5.60 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.                                     |

**57.4.5.61 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.                                     |

**57.4.5.62 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. |

**57.4.5.63 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.                                        |

**57.4.5.64 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.                                        |

**57.4.5.65 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.

**57.4.5.66 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 |

#### 57.4.5.67 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. |

#### 57.4.5.68 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. |

#### 57.4.5.69 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. |

**57.4.5.70** `status_t SAI_TransferTxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

**Deprecated** Do not use this function. It has been superceded by [SAI\\_TransferTxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

|                           |                                                                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                   |
| <i>handle</i>             | SAI handle pointer.                                                                                                                 |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                            |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |

## Returns

Status of this function. Return value is the status\_t.

**57.4.5.71** `status_t SAI_TransferRxSetFormat ( I2S_Type * base, sai_handle_t * handle, sai_transfer_format_t * format, uint32_t mclkSourceClockHz, uint32_t bclkSourceClockHz )`

**Deprecated** Do not use this function. It has been superceded by [SAI\\_TransferRxSetConfig](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.



## Parameters

|                           |                                                                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                   |
| <i>handle</i>             | SAI handle pointer.                                                                                                                 |
| <i>format</i>             | Pointer to the SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                            |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is a master clock, this value should equal the masterClockHz in format. |

## Returns

Status of this function. Return value is one of status\_t.

#### 57.4.5.72 status\_t SAI\_TransferSendNonBlocking ( I2S\_Type \* *base*, sai\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )

## Note

This API returns immediately after the transfer initiates. Call the SAI\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SAI\_Busy, the transfer is finished.

## Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | SAI base pointer.                                                      |
| <i>handle</i> | Pointer to the sai_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">sai_transfer_t</a> structure.               |

## Return values

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

#### 57.4.5.73 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.        |

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

## Parameters

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

## Return values

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

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

## Parameters

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

## Return values

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

**57.4.5.76 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. |

**57.4.5.77 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. |

**57.4.5.78 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. |

**57.4.5.79 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. |

**57.4.5.80 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. |

**57.4.5.81 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. |

## 57.5 SAI EDMA Driver

### 57.5.1 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.

#### Driver version

- #define [FSL\\_SAI\\_EDMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 5, 0))  
Version 2.5.0.

#### 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\\_TransferTxSetFormatEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Tx audio format.*
- void [SAI\\_TransferRxSetFormatEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_format\\_t](#) \*format, uint32\_t mclkSourceClockHz, uint32\_t bclkSourceClockHz)  
*Configures the SAI Rx audio format.*
- 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 DMA.*
- [status\\_t](#) [SAI\\_TransferReceiveEDMA](#) (I2S\_Type \*base, sai\_edma\_handle\_t \*handle, [sai\\_transfer\\_t](#) \*xfer)

- *Performs a non-blocking SAI receive using eDMA.*  
 • [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.*

## 57.5.2 Data Structure Documentation

### 57.5.2.1 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](#)  
*The transfer data count in a DMA request.*
- uint32\_t [state](#)  
*Internal state for SAI eDMA transfer.*
- [sai\\_edma\\_callback\\_t](#) [callback](#)  
*Callback for users while transfer finish or error occurs.*
- void \* [userData](#)

- *User callback parameter.*  
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.*
- 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

## 57.5.3 Function Documentation

**57.5.3.1 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. |

**57.5.3.2 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. |

**57.5.3.3 void SAI\_TransferTxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

**Deprecated** Do not use this function. It has been superceded by [SAI\\_TransferTxSetConfigEDMA](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

## Parameters

|                           |                                                                                                                                 |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                               |
| <i>handle</i>             | SAI eDMA handle pointer.                                                                                                        |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                     |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                        |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If bit clock source is master clock, this value should equals to masterClockHz in format. |

## Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

**57.5.3.4 void SAI\_TransferRxSetFormatEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_format\_t \* *format*, uint32\_t *mclkSourceClockHz*, uint32\_t *bclkSourceClockHz* )**

**Deprecated** Do not use this function. It has been superceded by [SAI\\_TransferRxSetConfigEDMA](#)

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred. This function also sets the eDMA parameter according to formatting requirements.

#### Parameters

|                           |                                                                                                                                      |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>               | SAI base pointer.                                                                                                                    |
| <i>handle</i>             | SAI eDMA handle pointer.                                                                                                             |
| <i>format</i>             | Pointer to SAI audio data format structure.                                                                                          |
| <i>mclkSource-ClockHz</i> | SAI master clock source frequency in Hz.                                                                                             |
| <i>bclkSource-ClockHz</i> | SAI bit clock source frequency in Hz. If a bit clock source is the master clock, this value should equal to masterClockHz in format. |

#### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Audio format set successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid. |

### 57.5.3.5 void SAI\_TransferTxSetConfigEDMA ( 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_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.      |

### 57.5.3.6 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.      |

### 57.5.3.7 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.           |

### 57.5.3.8 **status\_t SAI\_TransferReceiveEDMA ( I2S\_Type \* *base*, sai\_edma\_handle\_t \* *handle*, sai\_transfer\_t \* *xfer* )**

## Note

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

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. |
|-----------------------|-----------------------------|

### 57.5.3.9 **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.

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>loopTransfer-Count</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.      |

### 57.5.3.10 **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.

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 $\geq 1$ (loopTransferCount). |
| <i>loopTransfer-Count</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.         |

### 57.5.3.11 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. |

### 57.5.3.12 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. |

**57.5.3.13 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. |

#### 57.5.3.14 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. |

#### 57.5.3.15 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. |

#### 57.5.3.16 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. |

### 57.5.3.17 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 58

# SEMA4: Hardware Semaphores Driver

### 58.1 Overview

The MCUXpresso SDK provides a driver for the SEMA4 module of MCUXpresso SDK devices.

#### Macros

- #define [SEMA4\\_GATE\\_NUM\\_RESET\\_ALL](#) (64U)  
*The number to reset all SEMA4 gates.*
- #define [SEMA4\\_GATEn](#)(base, n) (((volatile uint8\_t \*)(&((base)->Gate00)))[(n)])  
*SEMA4 gate n register address.*

#### Functions

- void [SEMA4\\_Init](#) (SEMA4\_Type \*base)  
*Initializes the SEMA4 module.*
- void [SEMA4\\_Deinit](#) (SEMA4\_Type \*base)  
*De-initializes the SEMA4 module.*
- [status\\_t SEMA4\\_TryLock](#) (SEMA4\_Type \*base, uint8\_t gateNum, uint8\_t procNum)  
*Tries to lock the SEMA4 gate.*
- void [SEMA4\\_Lock](#) (SEMA4\_Type \*base, uint8\_t gateNum, uint8\_t procNum)  
*Locks the SEMA4 gate.*
- static void [SEMA4\\_Unlock](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Unlocks the SEMA4 gate.*
- static int32\_t [SEMA4\\_GetLockProc](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Gets the status of the SEMA4 gate.*
- [status\\_t SEMA4\\_ResetGate](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Resets the SEMA4 gate to an unlocked status.*
- static [status\\_t SEMA4\\_ResetAllGates](#) (SEMA4\_Type \*base)  
*Resets all SEMA4 gates to an unlocked status.*
- static void [SEMA4\\_EnableGateNotifyInterrupt](#) (SEMA4\_Type \*base, uint8\_t procNum, uint32\_t mask)  
*Enable the gate notification interrupt.*
- static void [SEMA4\\_DisableGateNotifyInterrupt](#) (SEMA4\_Type \*base, uint8\_t procNum, uint32\_t mask)  
*Disable the gate notification interrupt.*
- static uint32\_t [SEMA4\\_GetGateNotifyStatus](#) (SEMA4\_Type \*base, uint8\_t procNum)  
*Get the gate notification flags.*
- [status\\_t SEMA4\\_ResetGateNotify](#) (SEMA4\_Type \*base, uint8\_t gateNum)  
*Resets the SEMA4 gate IRQ notification.*
- static [status\\_t SEMA4\\_ResetAllGateNotify](#) (SEMA4\_Type \*base)  
*Resets all SEMA4 gates IRQ notification.*

## Driver version

- #define **FSL\_SEMA4\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 0, 2))  
*SEMA4 driver version.*

## 58.2 Macro Definition Documentation

### 58.2.1 #define SEMA4\_GATE\_NUM\_RESET\_ALL (64U)

## 58.3 Function Documentation

### 58.3.1 void SEMA4\_Init ( SEMA4\_Type \* *base* )

This function initializes the SEMA4 module. It only enables the clock but does not reset the gates because the module might be used by other processors at the same time. To reset the gates, call either SEMA4\_ResetGate or SEMA4\_ResetAllGates function.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SEMA4 peripheral base address. |
|-------------|--------------------------------|

### 58.3.2 void SEMA4\_Deinit ( SEMA4\_Type \* *base* )

This function de-initializes the SEMA4 module. It only disables the clock.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SEMA4 peripheral base address. |
|-------------|--------------------------------|

### 58.3.3 status\_t SEMA4\_TryLock ( SEMA4\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *procNum* )

This function tries to lock the specific SEMA4 gate. If the gate has been locked by another processor, this function returns an error code.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SEMA4 peripheral base address. |
|-------------|--------------------------------|

|                |                           |
|----------------|---------------------------|
| <i>gateNum</i> | Gate number to lock.      |
| <i>procNum</i> | Current processor number. |

Return values

|                        |                                                  |
|------------------------|--------------------------------------------------|
| <i>kStatus_Success</i> | Lock the sema4 gate successfully.                |
| <i>kStatus_Fail</i>    | Sema4 gate has been locked by another processor. |

#### 58.3.4 void SEMA4\_Lock ( SEMA4\_Type \* *base*, uint8\_t *gateNum*, uint8\_t *procNum* )

This function locks the specific SEMA4 gate. If the gate has been locked by other processors, this function waits until it is unlocked and then lock it.

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | SEMA4 peripheral base address. |
| <i>gateNum</i> | Gate number to lock.           |
| <i>procNum</i> | Current processor number.      |

#### 58.3.5 static void SEMA4\_Unlock ( SEMA4\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function unlocks the specific SEMA4 gate. It only writes unlock value to the SEMA4 gate register. However, it does not check whether the SEMA4 gate is locked by the current processor or not. As a result, if the SEMA4 gate is not locked by the current processor, this function has no effect.

Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | SEMA4 peripheral base address. |
| <i>gateNum</i> | Gate number to unlock.         |

#### 58.3.6 static int32\_t SEMA4\_GetLockProc ( SEMA4\_Type \* *base*, uint8\_t *gateNum* ) [inline], [static]

This function checks the lock status of a specific SEMA4 gate.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | SEMA4 peripheral base address. |
| <i>gateNum</i> | Gate number.                   |

## Returns

Return -1 if the gate is unlocked, otherwise return the processor number which has locked the gate.

### 58.3.7 **status\_t SEMA4\_ResetGate ( SEMA4\_Type \* *base*, uint8\_t *gateNum* )**

This function resets a SEMA4 gate to an unlocked status.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | SEMA4 peripheral base address. |
| <i>gateNum</i> | Gate number.                   |

## Return values

|                        |                                      |
|------------------------|--------------------------------------|
| <i>kStatus_Success</i> | SEMA4 gate is reset successfully.    |
| <i>kStatus_Fail</i>    | Some other reset process is ongoing. |

### 58.3.8 **static status\_t SEMA4\_ResetAllGates ( SEMA4\_Type \* *base* ) [inline], [static]**

This function resets all SEMA4 gate to an unlocked status.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SEMA4 peripheral base address. |
|-------------|--------------------------------|

## Return values

|                        |                              |
|------------------------|------------------------------|
| <i>kStatus_Success</i> | SEMA4 is reset successfully. |
|------------------------|------------------------------|

|                     |                                      |
|---------------------|--------------------------------------|
| <i>kStatus_Fail</i> | Some other reset process is ongoing. |
|---------------------|--------------------------------------|

### 58.3.9 static void SEMA4\_EnableGateNotifyInterrupt ( SEMA4\_Type \* *base*, uint8\_t *procNum*, uint32\_t *mask* ) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

|                |                                                                                      |
|----------------|--------------------------------------------------------------------------------------|
| <i>base</i>    | SEMA4 peripheral base address.                                                       |
| <i>procNum</i> | Current processor number.                                                            |
| <i>mask</i>    | OR'ed value of the gate index, for example: (1<<0)   (1<<1) means gate 0 and gate 1. |

### 58.3.10 static void SEMA4\_DisableGateNotifyInterrupt ( SEMA4\_Type \* *base*, uint8\_t *procNum*, uint32\_t *mask* ) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle.

Parameters

|                |                                                                                      |
|----------------|--------------------------------------------------------------------------------------|
| <i>base</i>    | SEMA4 peripheral base address.                                                       |
| <i>procNum</i> | Current processor number.                                                            |
| <i>mask</i>    | OR'ed value of the gate index, for example: (1<<0)   (1<<1) means gate 0 and gate 1. |

### 58.3.11 static uint32\_t SEMA4\_GetGateNotifyStatus ( SEMA4\_Type \* *base*, uint8\_t *procNum* ) [inline], [static]

Gate notification provides such feature, when core tried to lock the gate and failed, it could get notification when the gate is idle. The status flags are cleared automatically when the gate is locked by current core or locked again before the other core.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | SEMA4 peripheral base address. |
| <i>procNum</i> | Current processor number.      |

## Returns

OR'ed value of the gate index, for example:  $(1 \ll 0) \mid (1 \ll 1)$  means gate 0 and gate 1 flags are pending.

### 58.3.12 `status_t SEMA4_ResetGateNotify ( SEMA4_Type * base, uint8_t gateNum )`

This function resets a SEMA4 gate IRQ notification.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | SEMA4 peripheral base address. |
| <i>gateNum</i> | Gate number.                   |

## Return values

|                        |                                      |
|------------------------|--------------------------------------|
| <i>kStatus_Success</i> | Reset successfully.                  |
| <i>kStatus_Fail</i>    | Some other reset process is ongoing. |

### 58.3.13 `static status_t SEMA4_ResetAllGateNotify ( SEMA4_Type * base ) [inline], [static]`

This function resets all SEMA4 gate IRQ notifications.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SEMA4 peripheral base address. |
|-------------|--------------------------------|

## Return values

|                        |                                      |
|------------------------|--------------------------------------|
| <i>kStatus_Success</i> | Reset successfully.                  |
| <i>kStatus_Fail</i>    | Some other reset process is ongoing. |



## Chapter 59

# SEMC: Smart External DRAM Controller Driver

### 59.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Smart External DRAM Controller block of MCUXpresso SDK devices.

### 59.2 SEMC: Smart External DRAM Controller Driver

#### 59.2.1 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.

#### 59.2.2 SEMC Interrupt Operation

The interrupt and disable operation for SEMC.

#### 59.2.3 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.

### 59.3 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](#)

- *SEMC AXI queue a weight setting structure. [More...](#)*
- union [semc\\_queuea\\_weight\\_t](#)  
*SEMC AXI queue a weight setting union. [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_IOC_R_MUX_A8_SHIFT,
 kSEMC_MUXCSX0 = SEMC_IOC_R_MUX_CSX0_SHIFT,
 kSEMC_MUXCSX1 = SEMC_IOC_R_MUX_CSX1_SHIFT,
 kSEMC_MUXCSX2 = SEMC_IOC_R_MUX_CSX2_SHIFT,
 kSEMC_MUXCSX3 = SEMC_IOC_R_MUX_CSX3_SHIFT,
 kSEMC_MUXRDY = SEMC_IOC_R_MUX_RDY_SHIFT }
 SEMC IOMUXC.
• enum semc_iomux_nora27_pin {
 kSEMC_MORA27_NONE = 0,
 kSEMC_NORA27_MUXCSX3 = SEMC_IOC_R_MUX_CSX3_SHIFT,
 kSEMC_NORA27_MUXRDY = SEMC_IOC_R_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_Precharge,
 kSEMC_SDRAMCM_Prechargeall }
 SEMC IP command for SDARM.

```

## Driver version

- #define **FSL\_SEMC\_DRIVER\_VERSION** (MAKE\_VERSION(2, 4, 0))  
*SEMC driver version 2.3.1.*

## 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](#) type, 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.*

## 59.4 Data Structure Documentation

### 59.4.1 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.



- 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 `precalerIf precalerTimer_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 tAct2Prechage_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.

### 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

## 59.4.2 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 [tWehigh2Relow\\_Ns](#)  
*WE# high to RE# wait time: tWHR.*
- uint8\_t [tRehigh2Welow\\_Ns](#)  
*RE# high to WE# low wait time: tRHW.*
- uint8\_t [tAle2WriteStart\\_Ns](#)  
*ALE to write start wait time: tADL.*
- uint8\_t [tReady2Relow\\_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::tWehigh2Relow\_Ns

- (10) `uint8_t semc_nand_timing_config_t::tRehigh2Welow_Ns`
- (11) `uint8_t semc_nand_timing_config_t::tAle2WriteStart_Ns`
- (12) `uint8_t semc_nand_timing_config_t::tReady2Relow_Ns`
- (13) `uint8_t semc_nand_timing_config_t::tWehigh2Busy_Ns`

### 59.4.3 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 axiMemsizes_kbytes`  
*The memory size in unit of kbytes for AXI nand.*
- `uint32_t ipgAddress`  
*The base address for IPG nand .*
- `uint32_t ipgMemsizes_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 `kSEMC_MUXRDY` is not valid for CE pin setting.

- (2) `uint32_t semc_nand_config_t::axiAddress`
- (3) `uint32_t semc_nand_config_t::axiMemsizes_kbytes`
- (4) `uint32_t semc_nand_config_t::ipgAddress`
- (5) `uint32_t semc_nand_config_t::ipgMemsizes_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`

#### 59.4.4 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`  
*Wait ready polarity.*
- `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`

#### 59.4.5 struct `semc_sram_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_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.*
- `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) `uint8_t semc_sram_config_t::tCeSetup_Ns`
- (11) `uint8_t semc_sram_config_t::tCeHold_Ns`



- (12) `uint8_t semc_sram_config_t::tCeInterval_Ns`
- (13) `uint8_t semc_sram_config_t::readHoldTime_Ns`
- (14) `uint8_t semc_sram_config_t::tAddrSetup_Ns`
- (15) `uint8_t semc_sram_config_t::tAddrHold_Ns`
- (16) `uint8_t semc_sram_config_t::tWeLow_Ns`
- (17) `uint8_t semc_sram_config_t::tWeHigh_Ns`
- (18) `uint8_t semc_sram_config_t::tReLow_Ns`
- (19) `uint8_t semc_sram_config_t::tReHigh_Ns`
- (20) `uint8_t semc_sram_config_t::tTurnAround_Ns`
- (21) `uint8_t semc_sram_config_t::tAddr2WriteHold_Ns`
- (22) `uint8_t semc_sram_config_t::tWriteSetup_Ns`
- (23) `uint8_t semc_sram_config_t::tWriteHold_Ns`
- (24) `uint8_t semc_sram_config_t::latencyCount`
- (25) `uint8_t semc_sram_config_t::readCycle`
- (26) `uint8_t semc_sram_config_t::delayChain`

#### 59.4.6 struct `semc_dbi_config_t`

##### Data Fields

- `semc_iomux_pin csxPinMux`  
*The CE# pin mux.*
- `uint32_t address`  
*The base address.*
- `uint32_t memsize_kbytes`  
*The memory size in unit of 4kbytes.*
- `semc_dbi_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`

### 59.4.7 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 [slaveHitSwit](#): 8  
*weight of read/write switch for queue 0.*
- uint32\_t [slaveHitNoswitch](#): 8

*weight of read/write no 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::slaveHitSwith`
- (4) `uint32_t semc_queuea_weight_struct_t::slaveHitNoswitch`

### 59.4.8 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`

### 59.4.9 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 slaveHitSwith`: 8  
*weight of read/write switch for queue 1.*
- `uint32_t weightPagehit`: 8  
*weight of page hit for queue 1 only .*
- `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::slaveHitSwith`

(4) `uint32_t semc_queueb_weight_struct_t::weightPagehit`

(5) `uint32_t semc_queueb_weight_struct_t::bankRotation`

#### 59.4.10 `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`

#### 59.4.11 `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`

#### 59.4.12 `struct semc_config_t`

`busTimeoutCycles`: when `busTimeoutCycles` is zero, the bus timeout cycle is  $255 \times 1024$ . otherwise the bus timeout cycles is `busTimeoutCycles` $\times 1024$ . `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`

## 59.5 Macro Definition Documentation

### 59.5.1 `#define FSL_SEMC_DRIVER_VERSION (MAKE_VERSION(2, 4, 0))`

## 59.6 Enumeration Type Documentation

### 59.6.1 `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.

### 59.6.2 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.

### 59.6.3 enum semc\_waitready\_polarity\_t

Enumerator

*kSEMC\_LowActive* Low active.  
*kSEMC\_HighActive* High active.

### 59.6.4 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.

### 59.6.5 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.

### 59.6.6 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.

### 59.6.7 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.

### 59.6.8 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.

### 59.6.9 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.

### 59.6.10 enum semc\_caslatency\_t

Enumerator

*kSEMC\_LatencyOne* Latency 1.  
*kSEMC\_LatencyTwo* Latency 2.  
*kSEMC\_LatencyThree* Latency 3.

### 59.6.11 enum semc\_sdram\_column\_bit\_num\_t

Enumerator

*kSEMC\_SdramColumnm\_12bit* 12 bit.  
*kSEMC\_SdramColumnm\_11bit* 11 bit.  
*kSEMC\_SdramColumnm\_10bit* 10 bit.

*kSEMC\_SdramColumn\_9bit* 9 bit.  
*kSEMC\_SdramColumn\_8bit* 8 bit.

### 59.6.12 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.

### 59.6.13 enum semc\_nand\_column\_bit\_num\_t

Enumerator

*kSEMC\_NandColumn\_16bit* 16 bit.  
*kSEMC\_NandColumn\_15bit* 15 bit.  
*kSEMC\_NandColumn\_14bit* 14 bit.  
*kSEMC\_NandColumn\_13bit* 13 bit.  
*kSEMC\_NandColumn\_12bit* 12 bit.  
*kSEMC\_NandColumn\_11bit* 11 bit.  
*kSEMC\_NandColumn\_10bit* 10 bit.  
*kSEMC\_NandColumn\_9bit* 9 bit.

### 59.6.14 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.



### 59.6.15 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.

### 59.6.16 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.

### 59.6.17 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.

### 59.6.18 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.

### 59.6.19 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.

### 59.6.20 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.

### 59.6.21 enum smec\_port\_size\_t

Enumerator

*kSEMC\_PortSize8Bit* 8-Bit port size.  
*kSEMC\_PortSize16Bit* 16-Bit port size.  
*kSEMC\_PortSize32Bit* 32-Bit port size.

### 59.6.22 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.

### 59.6.23 enum semc\_dqs\_mode\_t

Enumerator

*kSEMC\_Loopbackinternal* Dummy read strobe loopbacked internally.  
*kSEMC\_Loopbackdqspad* Dummy read strobe loopbacked from DQS pad.

### 59.6.24 enum semc\_adv\_polarity\_t

Enumerator

*kSEMC\_AdvActiveLow* Adv active low.  
*kSEMC\_AdvActiveHigh* Adv active high.

### 59.6.25 enum semc\_sync\_mode\_t

Enumerator

*kSEMC\_AsyncMode* Async mode.  
*kSEMC\_SyncMode* Sync mode.

### 59.6.26 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.

**59.6.27 enum semc\_rdy\_polarity\_t**

Enumerator

*kSEMC\_RdyActiveLow* Adv active low.*kSEMC\_RdyActivehigh* Adv active low.**59.6.28 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).**59.6.29 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.**59.6.30 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.

### 59.6.31 enum semc\_ipcmd\_nor\_dbi\_t

Enumerator

*kSEMC\_NORDBICM\_Read* NOR read.

*kSEMC\_NORDBICM\_Write* NOR write.

### 59.6.32 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.

### 59.6.33 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\_Precharge* SDRAM precharge.

*kSEMC\_SDRAMCM\_Prechargeall* SDRAM precharge all.

## 59.7 Function Documentation

### 59.7.1 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. |
|---------------|-------------------------------------------|

**59.7.2 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. |

**59.7.3 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. |
|-------------|-------------------------------|

**59.7.4 status\_t SEMC\_ConfigureSDRAM ( SEMC\_Type \* *base*, semc\_sdram\_cs\_t *cs*, semc\_sdram\_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. |
|------------------|---------------------------|

**59.7.5** `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.     |

**59.7.6** `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.     |

**59.7.7** `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.     |

### 59.7.8 `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.     |

### 59.7.9 `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.     |

### 59.7.10 `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 :: <code>semc_interrupt_enable_t</code> . |

### 59.7.11 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 :: <code>semc_interrupt_enable_t</code> . |

### 59.7.12 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.

### 59.7.13 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 <a href="#">semc_interrupt_enable_t</a> . |

**59.7.14 static bool SEMC\_IsInIdle ( SEMC\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

True SEMC is in idle, false is not in idle.

**59.7.15 status\_t SEMC\_SendIPCommand ( SEMC\_Type \* *base*, semc\_mem\_type\_t *type*, uint32\_t *address*, uint32\_t *command*, uint32\_t *write*, uint32\_t \* *read* )**

## Parameters

|                |                                                                                                                                                                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | SEMC peripheral base address.                                                                                                                                                                                                                                                  |
| <i>type</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_dbi_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.                                                                                                                                                                                                                                                |

**59.7.16 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".  |

### 59.7.17 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.

### 59.7.18 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.                  |

### 59.7.19 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.                    |

**59.7.20** `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.                  |

**59.7.21** `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 60

## Smart Card

### 60.1 Overview

The MCUXpresso SDK provides a peripheral drivers for the UART-ISO7816 and EMVSIM modules of MCUXpresso SDK devices.

Smart Card driver provides the necessary functions to access and control integrated circuit cards. The driver controls communication modules (UART/EMVSIM) and handles special ICC sequences, such as the activation/deactivation (using EMVSIM IP or external interface chip). The Smart Card driver consists of two IPs (SmartCard\_Uart and SmartCard\_EmvSim drivers) and three PHY drivers (smartcard\_phy\_emvsim, smartcard\_phy\_tda8035, and smartcard\_phy\_gpio drivers). These drivers can be combined, which means that the Smart Card driver wraps one IP (transmission) and one PHY (interface) driver.

The driver provides asynchronous functions to communicate with the Integrated Circuit Card (ICC). The driver contains RTOS adaptation layers which use semaphores as synchronization objects of synchronous transfers. The RTOS driver support also provides protection for multithreading.

### 60.2 SmartCard Driver Initialization

The Smart Card Driver is initialized by calling the SMARTCARD\_Init() and [SMARTCARD\\_PHY\\_Init\(\)](#) functions. The Smart Card Driver initialization configuration structure requires these settings:

- Smart Card voltage class
- Smart Card Interface options such as the RST, IRQ, CLK pins, and so on.

The driver also supports user callbacks for assertion/de-assertion Smart Card events and transfer finish event. This feature is useful to detect the card presence or for handling transfer events, for example, in RTOS. The user should initialize the Smart Card driver, which consist of IP and PHY drivers.

### 60.3 SmartCard Call diagram

Because the call diagram is complex, the detailed use of the Smart Card driver is not described in this section. For details about using the Smart Card driver, see the Smart Card driver example which describes a simple use case.

### PHY driver

The Smart Card interface driver is initialized by calling the function [SMARTCARD\\_PHY\\_Init\(\)](#). During the initialization phase, Smart Card clock is configured and all hardware pins for IC handling are configured.

### Modules

- [Smart Card EMVSIM Driver](#)

- [Smart Card PHY Driver](#)
- [Smart Card PHY EMV SIM Driver](#)
- [Smart Card PHY GPIO Driver](#)
- [Smart Card PHY TDA8035 Driver](#)
- [Smart Card UART Driver](#)

## Data Structures

- struct [smartcard\\_card\\_params\\_t](#)  
*Defines card-specific parameters for Smart card driver. [More...](#)*
- struct [smartcard\\_timers\\_state\\_t](#)  
*Smart card defines the state of the EMV timers in the Smart card driver. [More...](#)*
- struct [smartcard\\_interface\\_config\\_t](#)  
*Defines user specified configuration of Smart card interface. [More...](#)*
- struct [smartcard\\_xfer\\_t](#)  
*Defines user transfer structure used to initialize transfer. [More...](#)*
- struct [smartcard\\_context\\_t](#)  
*Runtime state of the Smart card driver. [More...](#)*

## Macros

- #define [SMARTCARD\\_INIT\\_DELAY\\_CLOCK\\_CYCLES](#) (42000u)  
*Smart card global define which specify number of clock cycles until initial 'TS' character has to be received.*
- #define [SMARTCARD\\_EMV\\_ATR\\_DURATION\\_ETU](#) (20150u)  
*Smart card global define which specify number of clock cycles during which ATR string has to be received.*
- #define [SMARTCARD\\_TS\\_DIRECT\\_CONVENTION](#) (0x3Bu)  
*Smart card specification initial TS character definition of direct convention.*
- #define [SMARTCARD\\_TS\\_INVERSE\\_CONVENTION](#) (0x3Fu)  
*Smart card specification initial TS character definition of inverse convention.*

## Typedefs

- typedef void(\* [smartcard\\_interface\\_callback\\_t](#))(void \*smartcardContext, void \*param)  
*Smart card interface interrupt callback function type.*
- typedef void(\* [smartcard\\_transfer\\_callback\\_t](#))(void \*smartcardContext, void \*param)  
*Smart card transfer interrupt callback function type.*
- typedef void(\* [smartcard\\_time\\_delay\\_t](#))(uint32\_t us)  
*Time Delay function used to passive waiting using RTOS [us].*

## Enumerations

- enum {  
[kStatus\\_SMARTCARD\\_Success](#) = MAKE\_STATUS(kStatusGroup\_SMARTCARD, 0),  
[kStatus\\_SMARTCARD\\_TxBusy](#) = MAKE\_STATUS(kStatusGroup\_SMARTCARD, 1),  
[kStatus\\_SMARTCARD\\_RxBusy](#) = MAKE\_STATUS(kStatusGroup\_SMARTCARD, 2),  
[kStatus\\_SMARTCARD\\_NoTransferInProgress](#) = MAKE\_STATUS(kStatusGroup\_SMARTCAR-

```

D, 3),
kStatus_SMARTCARD_Timeout = MAKE_STATUS(kStatusGroup_SMARTCARD, 4),
kStatus_SMARTCARD_Initialized,
kStatus_SMARTCARD_PhyInitialized,
kStatus_SMARTCARD_CardNotActivated = MAKE_STATUS(kStatusGroup_SMARTCARD, 7),
kStatus_SMARTCARD_InvalidInput,
kStatus_SMARTCARD_OtherError = MAKE_STATUS(kStatusGroup_SMARTCARD, 9) }

```

*Smart card Error codes.*

- enum [smartcard\\_control\\_t](#)  
*Control codes for the Smart card protocol timers and misc.*
- enum [smartcard\\_card\\_voltage\\_class\\_t](#)  
*Defines Smart card interface voltage class values.*
- enum [smartcard\\_transfer\\_state\\_t](#)  
*Defines Smart card I/O transfer states.*
- enum [smartcard\\_reset\\_type\\_t](#)  
*Defines Smart card reset types.*
- enum [smartcard\\_transport\\_type\\_t](#)  
*Defines Smart card transport protocol types.*
- enum [smartcard\\_parity\\_type\\_t](#)  
*Defines Smart card data parity types.*
- enum [smartcard\\_card\\_convention\\_t](#)  
*Defines data Convention format.*
- enum [smartcard\\_interface\\_control\\_t](#)  
*Defines Smart card interface IC control types.*
- enum [smartcard\\_direction\\_t](#)  
*Defines transfer direction.*

## Driver version

- #define [FSL\\_SMARTCARD\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 2, 2))  
*Smart card driver version 2.2.2.*

## 60.4 Data Structure Documentation

### 60.4.1 struct smartcard\_card\_params\_t

#### Data Fields

- uint16\_t [Fi](#)  
*4 bits Fi - clock rate conversion integer*
- uint8\_t [fMax](#)  
*Maximum Smart card frequency in MHz.*
- uint8\_t [WI](#)  
*8 bits WI - work wait time integer*
- uint8\_t [Di](#)  
*4 bits DI - baud rate divisor*
- uint8\_t [BWI](#)  
*4 bits BWI - block wait time integer*
- uint8\_t [CWI](#)



- *4 bits CWI - character wait time integer*
- uint8\_t [BGI](#)
- *4 bits BGI - block guard time integer*
- uint8\_t [GTN](#)
- *8 bits GTN - extended guard time integer*
- uint8\_t [IFSC](#)
- *Indicates IFSC value of the card.*
- uint8\_t [modeNegotiable](#)
- *Indicates if the card acts in negotiable or a specific mode.*
- uint8\_t [currentD](#)
- *4 bits DI - current baud rate divisor*
- uint8\_t [status](#)
- *Indicates smart card status.*
- bool [t0Indicated](#)
- *Indicates ff T=0 indicated in TD1 byte.*
- bool [t1Indicated](#)
- *Indicates if T=1 indicated in TD2 byte.*
- bool [atrComplete](#)
- *Indicates whether the ATR received from the card was complete or not.*
- bool [atrValid](#)
- *Indicates whether the ATR received from the card was valid or not.*
- bool [present](#)
- *Indicates if a smart card is present.*
- bool [active](#)
- *Indicates if the smart card is activated.*
- bool [faulty](#)
- *Indicates whether smart card/interface is faulty.*
- [smartcard\\_card\\_convention\\_t convention](#)
- *Card convention, kSMARTCARD\_DirectConvention for direct convention, kSMARTCARD\_InverseConvention for inverse convention.*

## Field Documentation

(1) uint8\_t smartcard\_card\_params\_t::modeNegotiable

### 60.4.2 struct smartcard\_timers\_state\_t

#### Data Fields

- volatile bool [adtExpired](#)
- *Indicates whether ADT timer expired.*
- volatile bool [wwtExpired](#)
- *Indicates whether WWT timer expired.*
- volatile bool [cwtExpired](#)
- *Indicates whether CWT timer expired.*
- volatile bool [bwtExpired](#)
- *Indicates whether BWT timer expired.*
- volatile bool [initCharTimerExpired](#)
- *Indicates whether reception timer for initialization character (TS) after the RST has expired*

### 60.4.3 struct smartcard\_interface\_config\_t

#### Data Fields

- uint32\_t [smartCardClock](#)  
*Smart card interface clock [Hz].*
- uint32\_t [clockToResetDelay](#)  
*Indicates clock to RST apply delay [smart card clock cycles].*
- uint8\_t [clockModule](#)  
*Smart card clock module number.*
- uint8\_t [clockModuleChannel](#)  
*Smart card clock module channel number.*
- uint8\_t [clockModuleSourceClock](#)  
*Smart card clock module source clock [e.g., BusClk].*
- [smartcard\\_card\\_voltage\\_class\\_t](#) [vcc](#)  
*Smart card voltage class.*
- uint8\_t [controlPort](#)  
*Smart card PHY control port instance.*
- uint8\_t [controlPin](#)  
*Smart card PHY control pin instance.*
- uint8\_t [irqPort](#)  
*Smart card PHY Interrupt port instance.*
- uint8\_t [irqPin](#)  
*Smart card PHY Interrupt pin instance.*
- uint8\_t [resetPort](#)  
*Smart card reset port instance.*
- uint8\_t [resetPin](#)  
*Smart card reset pin instance.*
- uint8\_t [vsel0Port](#)  
*Smart card PHY Vsel0 control port instance.*
- uint8\_t [vsel0Pin](#)  
*Smart card PHY Vsel0 control pin instance.*
- uint8\_t [vsel1Port](#)  
*Smart card PHY Vsel1 control port instance.*
- uint8\_t [vsel1Pin](#)  
*Smart card PHY Vsel1 control pin instance.*
- uint8\_t [dataPort](#)  
*Smart card PHY data port instance.*
- uint8\_t [dataPin](#)  
*Smart card PHY data pin instance.*
- uint8\_t [dataPinMux](#)  
*Smart card PHY data pin mux option.*
- uint8\_t [tsTimerId](#)  
*Numerical identifier of the External HW timer for Initial character detection.*

#### 60.4.4 struct smartcard\_xfer\_t

##### Data Fields

- [smartcard\\_direction\\_t direction](#)  
*Direction of communication.*
- `uint8_t * buff`  
*The buffer of data.*
- `size_t size`  
*The number of transferred units.*

##### Field Documentation

##### (1) smartcard\_direction\_t smartcard\_xfer\_t::direction

(RX/TX)

##### (2) uint8\_t\* smartcard\_xfer\_t::buff

##### (3) size\_t smartcard\_xfer\_t::size

#### 60.4.5 struct smartcard\_context\_t

##### Data Fields

- `void * base`  
*Smart card module base address.*
- [smartcard\\_direction\\_t direction](#)  
*Direction of communication.*
- `uint8_t * xBuff`  
*The buffer of data being transferred.*
- `volatile size_t xSize`  
*The number of bytes to be transferred.*
- `volatile bool xIsBusy`  
*True if there is an active transfer.*
- `uint8_t txFifoEntryCount`  
*Number of data word entries in transmit FIFO.*
- `uint8_t rxFifoThreshold`  
*The max value of the receiver FIFO threshold.*
- [smartcard\\_interface\\_callback\\_t interfaceCallback](#)  
*Callback to invoke after interface IC raised interrupt.*
- [smartcard\\_transfer\\_callback\\_t transferCallback](#)  
*Callback to invoke after transfer event occur.*
- `void * interfaceCallbackParam`  
*Interface callback parameter pointer.*
- `void * transferCallbackParam`  
*Transfer callback parameter pointer.*
- [smartcard\\_time\\_delay\\_t timeDelay](#)  
*Function which handles time delay defined by user or RTOS.*

- [smartcard\\_reset\\_type\\_t resetType](#)  
*Indicates whether a Cold reset or Warm reset was requested.*
- [smartcard\\_transport\\_type\\_t tType](#)  
*Indicates current transfer protocol (T0 or T1)*
- volatile [smartcard\\_transfer\\_state\\_t transferState](#)  
*Indicates the current transfer state.*
- [smartcard\\_timers\\_state\\_t timersState](#)  
*Indicates the state of different protocol timers used in driver.*
- [smartcard\\_card\\_params\\_t cardParams](#)  
*Smart card parameters(ATR and current) and interface slots states(ATR and current)*
- uint8\_t [IFSD](#)  
*Indicates the terminal IFSD.*
- [smartcard\\_parity\\_type\\_t parity](#)  
*Indicates current parity even/odd.*
- volatile bool [rxtCrossed](#)  
*Indicates whether RXT thresholds has been crossed.*
- volatile bool [txtCrossed](#)  
*Indicates whether TXT thresholds has been crossed.*
- volatile bool [wtxRequested](#)  
*Indicates whether WTX has been requested or not.*
- volatile bool [parityError](#)  
*Indicates whether a parity error has been detected.*
- uint8\_t [statusBytes](#) [2]  
*Used to store Status bytes SW1, SW2 of the last executed card command response.*
- [smartcard\\_interface\\_config\\_t interfaceConfig](#)  
*Smart card interface configuration structure.*
- bool [abortTransfer](#)  
*Used to abort transfer.*

## Field Documentation

- (1) **smartcard\_direction\_t smartcard\_context\_t::direction**  
(RX/TX)
- (2) **uint8\_t\* smartcard\_context\_t::xBuff**
- (3) **volatile size\_t smartcard\_context\_t::xSize**
- (4) **volatile bool smartcard\_context\_t::xIsBusy**
- (5) **uint8\_t smartcard\_context\_t::txFifoEntryCount**
- (6) **uint8\_t smartcard\_context\_t::rxFifoThreshold**
- (7) **smartcard\_interface\_callback\_t smartcard\_context\_t::interfaceCallback**
- (8) **smartcard\_transfer\_callback\_t smartcard\_context\_t::transferCallback**
- (9) **void\* smartcard\_context\_t::interfaceCallbackParam**

- (10) `void* smartcard_context_t::transferCallbackParam`
- (11) `smartcard_time_delay_t smartcard_context_t::timeDelay`
- (12) `smartcard_reset_type_t smartcard_context_t::resetType`
- (13) `bool smartcard_context_t::abortTransfer`

## 60.5 Enumeration Type Documentation

### 60.5.1 anonymous enum

Enumerator

*kStatus\_SMARTCARD\_Success* Transfer ends successfully.  
*kStatus\_SMARTCARD\_TxBusy* Transmit in progress.  
*kStatus\_SMARTCARD\_RxBusy* Receiving in progress.  
*kStatus\_SMARTCARD\_NoTransferInProgress* No transfer in progress.  
*kStatus\_SMARTCARD\_Timeout* Transfer ends with time-out.  
*kStatus\_SMARTCARD\_Initialized* Smart card driver is already initialized.  
*kStatus\_SMARTCARD\_PhyInitialized* Smart card PHY drive is already initialized.  
*kStatus\_SMARTCARD\_CardNotActivated* Smart card is not activated.  
*kStatus\_SMARTCARD\_InvalidInput* Function called with invalid input arguments.  
*kStatus\_SMARTCARD\_OtherError* Some other error occur.

### 60.5.2 enum smartcard\_control\_t

### 60.5.3 enum smartcard\_direction\_t

## **60.6 Smart Card PHY TDA8035 Driver**

The Smart Card interface TDA8035 driver handles the external interface chip TDA8035 which supports all necessary functions to control the ICC. These functions involve PHY pin initialization, ICC voltage selection and activation, ICC clock generation, ICC card detection, and activation/deactivation sequences.

## **60.7 Smart Card PHY EMVSIM Driver**

The Smart Card interface EMVSIM driver handles the EMVSIM peripheral, which covers all necessary functions to control the ICC. These functions are ICC clock setup, ICC voltage turning on/off, ICC card detection, activation/deactivation, and ICC reset sequences. The EMVSIM peripheral covers all features of interface ICC chips.

## **60.8 Smart Card PHY GPIO Driver**

The Smart Card interface GPIO driver handles the GPIO and FTM/TPM peripheral for clock generation, which covers all necessary functions to control the ICC. These functions are ICC clock setup, ICC voltage turning on/off, activation/deactivation, and ICC reset sequences. This driver doesn't support the ICC pin short circuit protection and an emergency deactivation.



## **60.9 Smart Card UART Driver**

The Smart Card UART driver uses a standard UART peripheral which supports the ISO-7816 standard. The driver supports transmission functionality in the CPU mode. The driver also supports non-blocking (asynchronous) type of data transfers. The blocking (synchronous) transfer is supported only by the RTOS adaptation layer.

## 60.10 Smart Card EMVSIM Driver

### 60.10.1 Overview

The SmartCard EMVSIM driver covers the transmission functionality in the CPU mode. The driver supports non-blocking (asynchronous) type of data transfers. The blocking (synchronous) transfer is supported only by the RTOS adaptation layer.

### Macros

- #define `SMARTCARD_EMV_RX_NACK_THRESHOLD` (5u)  
*EMV RX NACK interrupt generation threshold.*
- #define `SMARTCARD_EMV_TX_NACK_THRESHOLD` (5u)  
*EMV TX NACK interrupt generation threshold.*
- #define `SMARTCARD_WWT_ADJUSTMENT` (160u)  
*Smart card Word Wait Timer adjustment value.*
- #define `SMARTCARD_CWT_ADJUSTMENT` (3u)  
*Smart card Character Wait Timer adjustment value.*

### Enumerations

- enum `emvsim_gpc_clock_select_t` {  
    `kEMVSIM_GPCClockDisable` = 0u,  
    `kEMVSIM_GPCCardClock` = 1u,  
    `kEMVSIM_GPCRxClock` = 2u,  
    `kEMVSIM_GPCTxClock` = 3u }  
*General Purpose Counter clock selections.*
- enum `emvsim_presence_detect_edge_t` {  
    `kEMVSIM_DetectOnFallingEdge` = 0u,  
    `kEMVSIM_DetectOnRisingEdge` = 1u }  
*EMVSIM card presence detection edge control.*
- enum `emvsim_presence_detect_status_t` {  
    `kEMVSIM_DetectPinIsLow` = 0u,  
    `kEMVSIM_DetectPinIsHigh` = 1u }  
*EMVSIM card presence detection status.*

### Smart card EMVSIM Driver

- void `SMARTCARD_EMVSIM_GetDefaultConfig` (`smartcard_card_params_t` \*cardParams)  
*Fills in the smartcard\_card\_params structure with default values according to the EMV 4.3 specification.*
- `status_t` `SMARTCARD_EMVSIM_Init` (`EMVSIM_Type` \*base, `smartcard_context_t` \*context, `uint32_t` srcClock\_Hz)  
*Initializes an EMVSIM peripheral for the Smart card/ISO-7816 operation.*
- void `SMARTCARD_EMVSIM_Deinit` (`EMVSIM_Type` \*base)

*This function disables the EMVSIM interrupts, disables the transmitter and receiver, flushes the FIFOs, and gates EMVSIM clock in SIM.*

- `int32_t SMARTCARD_EMVSIM_GetTransferRemainingBytes` (EMVSIM\_Type \*base, `smartcard-_context_t` \*context)  
*Returns whether the previous EMVSIM transfer has finished.*
- `status_t SMARTCARD_EMVSIM_AbortTransfer` (EMVSIM\_Type \*base, `smartcard_context_t` \*context)  
*Terminates an asynchronous EMVSIM transfer early.*
- `status_t SMARTCARD_EMVSIM_TransferNonBlocking` (EMVSIM\_Type \*base, `smartcard-_context_t` \*context, `smartcard_xfer_t` \*xfer)  
*Transfer data using interrupts.*
- `status_t SMARTCARD_EMVSIM_Control` (EMVSIM\_Type \*base, `smartcard_context_t` \*context, `smartcard_control_t` control, `uint32_t` param)  
*Controls the EMVSIM module per different user request.*
- `void SMARTCARD_EMVSIM_IRQHandler` (EMVSIM\_Type \*base, `smartcard_context_t` \*context)  
*Handles EMVSIM module interrupts.*

## 60.10.2 Enumeration Type Documentation

### 60.10.2.1 enum emvsim\_gpc\_clock\_select\_t

Enumerator

***kEMVSIM\_GPCClockDisable*** Disabled.  
***kEMVSIM\_GPCCardClock*** Card clock.  
***kEMVSIM\_GPCRxClock*** Receive clock.  
***kEMVSIM\_GPCTxClock*** Transmit ETU clock.

### 60.10.2.2 enum emvsim\_presence\_detect\_edge\_t

Enumerator

***kEMVSIM\_DetectOnFallingEdge*** Presence detected on the falling edge.  
***kEMVSIM\_DetectOnRisingEdge*** Presence detected on the rising edge.

### 60.10.2.3 enum emvsim\_presence\_detect\_status\_t

Enumerator

***kEMVSIM\_DetectPinIsLow*** Presence detected pin is logic low.  
***kEMVSIM\_DetectPinIsHigh*** Presence detected pin is logic high.

### 60.10.3 Function Documentation

60.10.3.1 void SMARTCARD\_EMV SIM\_GetDefaultConfig ( smartcard\_card\_params\_t \*  
*cardParams* )

## Parameters

|                   |                                                                                                                                                                                          |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>cardParams</i> | The configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: Fi = 372; Di = 1; currentD = 1; WI = 0x0A; GTN = 0x00; with default values. |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 60.10.3.2 status\_t SMARTCARD\_EMVSIM\_Init ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context*, uint32\_t *srcClock\_Hz* )

This function un-gates the EMVSIM clock, initializes the module to EMV default settings, configures the IRQ, enables the module-level interrupt to the core and, initializes the driver context.

## Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>base</i>        | The EMVSIM peripheral base address.                   |
| <i>context</i>     | A pointer to the smart card driver context structure. |
| <i>srcClock_Hz</i> | Smart card clock generation module source clock.      |

## Returns

An error code or kStatus\_SMARTCARD\_Success.

### 60.10.3.3 void SMARTCARD\_EMVSIM\_Deinit ( EMVSIM\_Type \* *base* )

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The EMVSIM module base address. |
|-------------|---------------------------------|

### 60.10.3.4 int32\_t SMARTCARD\_EMVSIM\_GetTransferRemainingBytes ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context* )

When performing an async transfer, call this function to ascertain the context of the current transfer: in progress (or busy) or complete (success). If the transfer is still in progress, the user can obtain the number of words that have not been transferred.

## Parameters

---

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The EMVSIM module base address.                     |
| <i>context</i> | A pointer to a smart card driver context structure. |

Returns

The number of bytes not transferred.

#### 60.10.3.5 **status\_t SMARTCARD\_EMVSIM\_AbortTransfer ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context* )**

During an async EMVSIM transfer, the user can terminate the transfer early if the transfer is still in progress.

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The EMVSIM peripheral address.                      |
| <i>context</i> | A pointer to a smart card driver context structure. |

Return values

|                                                |                                           |
|------------------------------------------------|-------------------------------------------|
| <i>kStatus_SMARTCARD_-Success</i>              | The transmit abort was successful.        |
| <i>kStatus_SMARTCARD_-NoTransmitInProgress</i> | No transmission is currently in progress. |

#### 60.10.3.6 **status\_t SMARTCARD\_EMVSIM\_TransferNonBlocking ( EMVSIM\_Type \* *base*, smartcard\_context\_t \* *context*, smartcard\_xfer\_t \* *xfer* )**

A non-blocking (also known as asynchronous) function means that the function returns immediately after initiating the transfer function. The application has to get the transfer status to see when the transfer is complete. In other words, after calling the non-blocking (asynchronous) transfer function, the application must get the transfer status to check if the transmit is completed or not.

Parameters

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

|                |                                                                                               |
|----------------|-----------------------------------------------------------------------------------------------|
| <i>context</i> | A pointer to a smart card driver context structure.                                           |
| <i>xfer</i>    | A pointer to the smart card transfer structure where the linked buffers and sizes are stored. |

## Returns

An error code or kStatus\_SMARTCARD\_Success.

**60.10.3.7** `status_t SMARTCARD_EMV SIM_Control ( EMV SIM_Type * base,  
smartcard_context_t * context, smartcard_control_t control, uint32_t param )`

## Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The EMV SIM peripheral base address.                |
| <i>context</i> | A pointer to a smart card driver context structure. |
| <i>control</i> | Control type.                                       |
| <i>param</i>   | Integer value of specific to control command.       |

return kStatus\_SMARTCARD\_Success in success. return kStatus\_SMARTCARD\_OtherError in case of error.

**60.10.3.8** `void SMARTCARD_EMV SIM_IRQHandler ( EMV SIM_Type * base,  
smartcard_context_t * context )`

## Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The EMV SIM peripheral base address.                |
| <i>context</i> | A pointer to a smart card driver context structure. |

## 60.11 Smart Card PHY Driver

### 60.11.1 Overview

#### Macros

- #define `SMARTCARD_ATR_DURATION_ADJUSTMENT` (360u)  
*Smart card definition which specifies the adjustment number of clock cycles during which an ATR string has to be received.*
- #define `SMARTCARD_INIT_DELAY_CLOCK_CYCLES_ADJUSTMENT` (4200u)  
*Smart card definition which specifies the adjustment number of clock cycles until an initial 'TS' character has to be received.*

#### Functions

- void `SMARTCARD_PHY_GetDefaultConfig` (`smartcard_interface_config_t` \*config)  
*Fills in the configuration structure with default values.*
- `status_t SMARTCARD_PHY_Init` (void \*base, `smartcard_interface_config_t` const \*config, uint32\_t srcClock\_Hz)  
*Initializes a Smart card interface instance.*
- void `SMARTCARD_PHY_Deinit` (void \*base, `smartcard_interface_config_t` const \*config)  
*De-initializes a Smart card interface, stops the Smart card clock, and disables the VCC.*
- `status_t SMARTCARD_PHY_Activate` (void \*base, `smartcard_context_t` \*context, `smartcard_reset_type_t` resetType)  
*Activates the Smart card IC.*
- `status_t SMARTCARD_PHY_Deactivate` (void \*base, `smartcard_context_t` \*context)  
*De-activates the Smart card IC.*
- `status_t SMARTCARD_PHY_Control` (void \*base, `smartcard_context_t` \*context, `smartcard_interface_control_t` control, uint32\_t param)  
*Controls the Smart card interface IC.*

### 60.11.2 Macro Definition Documentation

#### 60.11.2.1 #define SMARTCARD\_INIT\_DELAY\_CLOCK\_CYCLES\_ADJUSTMENT (4200u)

### 60.11.3 Function Documentation

#### 60.11.3.1 void SMARTCARD\_PHY\_GetDefaultConfig ( smartcard\_interface\_config\_t \* config )



## Parameters

|               |                                                                                                                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>config</i> | The Smart card user configuration structure which contains configuration structure of type <a href="#">smartcard_interface_config_t</a> . Function fill in members: clockToResetDelay = 42000, vcc = kSmartcardVoltageClassB3_3V, with default values. |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 60.11.3.2 status\_t SMARTCARD\_PHY\_Init ( void \* *base*, smartcard\_interface\_config\_t const \* *config*, uint32\_t *srcClock\_Hz* )

## Parameters

|                    |                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | The Smart card peripheral base address.                                                                                                                                                         |
| <i>config</i>      | The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . Call the function <a href="#">SMARTCARD_PHY_GetDefaultConfig()</a> to fill the configuration structure. |
| <i>srcClock_Hz</i> | Smart card clock generation module source clock.                                                                                                                                                |

## Return values

|                                  |                                                   |
|----------------------------------|---------------------------------------------------|
| <i>kStatus_SMARTCARD_Success</i> | or kStatus_SMARTCARD_OtherError in case of error. |
|----------------------------------|---------------------------------------------------|

### 60.11.3.3 void SMARTCARD\_PHY\_Deinit ( void \* *base*, smartcard\_interface\_config\_t const \* *config* )

## Parameters

|               |                                                                                         |
|---------------|-----------------------------------------------------------------------------------------|
| <i>base</i>   | The Smart card peripheral module base address.                                          |
| <i>config</i> | The user configuration structure of type <a href="#">smartcard_interface_config_t</a> . |

### 60.11.3.4 status\_t SMARTCARD\_PHY\_Activate ( void \* *base*, smartcard\_context\_t \* *context*, smartcard\_reset\_type\_t *resetType* )

## Parameters

---

|                  |                                                                                            |
|------------------|--------------------------------------------------------------------------------------------|
| <i>base</i>      | The Smart card peripheral module base address.                                             |
| <i>context</i>   | A pointer to a Smart card driver context structure.                                        |
| <i>resetType</i> | type of reset to be performed, possible values = kSmartcardColdReset, kSmartcard-WarmReset |

Return values

|                                   |                                                   |
|-----------------------------------|---------------------------------------------------|
| <i>kStatus_SMARTCARD_-Success</i> | or kStatus_SMARTCARD_OtherError in case of error. |
|-----------------------------------|---------------------------------------------------|

#### 60.11.3.5 status\_t SMARTCARD\_PHY\_Deactivate ( void \* *base*, smartcard\_context\_t \* *context* )

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The Smart card peripheral module base address.      |
| <i>context</i> | A pointer to a Smart card driver context structure. |

Return values

|                                   |                                                   |
|-----------------------------------|---------------------------------------------------|
| <i>kStatus_SMARTCARD_-Success</i> | or kStatus_SMARTCARD_OtherError in case of error. |
|-----------------------------------|---------------------------------------------------|

#### 60.11.3.6 status\_t SMARTCARD\_PHY\_Control ( void \* *base*, smartcard\_context\_t \* *context*, smartcard\_interface\_control\_t *control*, uint32\_t *param* )

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>base</i>    | The Smart card peripheral module base address.      |
| <i>context</i> | A pointer to a Smart card driver context structure. |
| <i>control</i> | A interface command type.                           |
| <i>param</i>   | Integer value specific to control type              |

Return values

|                                        |                                                   |
|----------------------------------------|---------------------------------------------------|
| <i>kStatus_SMARTCARD_-<br/>Success</i> | or kStatus_SMARTCARD_OtherError in case of error. |
|----------------------------------------|---------------------------------------------------|

## Chapter 61

# SNVS: Secure Non-Volatile Storage

### 61.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage (SNVS) module.

The SNVS module is designed to safely hold security-related data such as cryptographic key, time counter, monotonic counter, and general purpose security information. The SNVS includes a low power section, namely SNVS\_LP, that is battery backed by the SVNS (or VBAT) power domain. This enables it to keep this data valid and continue to increment the time counter when the power goes down in the rest of the SoC. The always-powered-up part of the module is isolated from the rest of the logic to ensure that it does not get corrupted when the SoC is powered down. The SNVS is designed to comply with Digital Rights Management (DRM) and other security application rules and requirements. This trusted hardware provides features that allow the system software designer to ensure that the data kept by the device is certifiable. Specially, it incorporates a security monitor that checks for various security conditions. If a security violation is indicated then it invalidates access to its sensitive data, and the secret data, for instance, Zeroizable Secret Key, is zeroized. the SNVS can be also configured to bypass its security features and protection mechanism. In this case it can be used by systems that do not require security.

### Modules

- [Secure Non-Volatile Storage High-Power](#)
- [Secure Non-Volatile Storage Low-Power](#)

## 61.2 Secure Non-Volatile Storage High-Power

### 61.2.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the Secure Non-Volatile Storage High-Power(S-NVS-HP) module.

The SNVS\_HP is in the chip's power-supply domain and thus receives the power along with the rest of the chip. The SNVS\_HP provides an interface between the SNVS\_LP and the rest of the system; there is no way to access the SNVS\_LP registers except through the SNVS\_HP. For access to the SNVS\_LP registers, the SNVS\_HP must be powered up. It uses a register access permission policy to determine whether the access to the particular registers is permitted.

### Data Structures

- struct [snvs\\_hp\\_rtc\\_datetime\\_t](#)  
*Structure is used to hold the date and time. [More...](#)*
- struct [snvs\\_hp\\_rtc\\_config\\_t](#)  
*SNVS config structure. [More...](#)*

### Macros

- #define [SNVS\\_MAKE\\_HP\\_SV\\_FLAG](#)(x) (1U << (SNVS\_HPSVSR\_SV0\_SHIFT + (x)))  
*Macro to make security violation flag.*

### Enumerations

- enum [snvs\\_hp\\_interrupts\\_t](#) {  
  [kSNVS\\_RTC\\_AlarmInterrupt](#) = SNVS\_HPCR\_HPTA\_EN\_MASK,  
  [kSNVS\\_RTC\\_PeriodicInterrupt](#) = SNVS\_HPCR\_PI\_EN\_MASK }  
*List of SNVS interrupts.*
- enum [snvs\\_hp\\_status\\_flags\\_t](#) {  
  [kSNVS\\_RTC\\_AlarmInterruptFlag](#) = SNVS\_HPSR\_HPTA\_MASK,  
  [kSNVS\\_RTC\\_PeriodicInterruptFlag](#) = SNVS\_HPSR\_PI\_MASK,  
  [kSNVS\\_ZMK\\_ZeroFlag](#) = (int)SNVS\_HPSR\_ZMK\_ZERO\_MASK,  
  [kSNVS\\_OTPMK\\_ZeroFlag](#) = SNVS\_HPSR\_OTPMK\_ZERO\_MASK }  
*List of SNVS flags.*
- enum [snvs\\_hp\\_sv\\_status\\_flags\\_t](#) {

```

kSNVS_LP_ViolationFlag = SNVS_HPSVSR_SW_LPSV_MASK,
kSNVS_ZMK_EccFailFlag = SNVS_HPSVSR_ZMK_ECC_FAIL_MASK,
kSNVS_LP_SoftwareViolationFlag = SNVS_HPSVSR_SW_LPSV_MASK,
kSNVS_FatalSoftwareViolationFlag = SNVS_HPSVSR_SW_FSV_MASK,
kSNVS_SoftwareViolationFlag = SNVS_HPSVSR_SW_SV_MASK,
kSNVS_Violation0Flag = SNVS_HPSVSR_SV0_MASK,
kSNVS_Violation1Flag = SNVS_HPSVSR_SV1_MASK,
kSNVS_Violation2Flag = SNVS_HPSVSR_SV2_MASK,
kSNVS_Violation4Flag = SNVS_HPSVSR_SV4_MASK,
kSNVS_Violation5Flag = SNVS_HPSVSR_SV5_MASK }

```

*List of SNVS security violation flags.*

- enum `snvs_hp_ssm_state_t` {  
`kSNVS_SSMInit` = 0x00,  
`kSNVS_SSMHardFail` = 0x01,  
`kSNVS_SSMSoftFail` = 0x03,  
`kSNVS_SSMInitInter` = 0x08,  
`kSNVS_SSMCheck` = 0x09,  
`kSNVS_SSMNonSecure` = 0x0B,  
`kSNVS_SSMTrusted` = 0x0D,  
`kSNVS_SSMSecure` = 0x0F }

*List of SNVS Security State Machine State.*

## Functions

- static void `SNVS_HP_EnableMasterKeySelection` (SNVS\_Type \*base, bool enable)  
*Enable or disable master key selection.*
- static void `SNVS_HP_ProgramZeroizableMasterKey` (SNVS\_Type \*base)  
*Trigger to program Zeroizable Master Key.*
- static void `SNVS_HP_ChangeSSMState` (SNVS\_Type \*base)  
*Trigger SSM State Transition.*
- static void `SNVS_HP_SetSoftwareFatalSecurityViolation` (SNVS\_Type \*base)  
*Trigger Software Fatal Security Violation.*
- static void `SNVS_HP_SetSoftwareSecurityViolation` (SNVS\_Type \*base)  
*Trigger Software Security Violation.*
- static `snvs_hp_ssm_state_t` `SNVS_HP_GetSSMState` (SNVS\_Type \*base)  
*Get current SSM State.*
- static void `SNVS_HP_ResetLP` (SNVS\_Type \*base)  
*Reset the SNVS LP section.*
- static uint32\_t `SNVS_HP_GetStatusFlags` (SNVS\_Type \*base)  
*Get the SNVS HP status flags.*
- static void `SNVS_HP_ClearStatusFlags` (SNVS\_Type \*base, uint32\_t mask)  
*Clear the SNVS HP status flags.*
- static uint32\_t `SNVS_HP_GetSecurityViolationStatusFlags` (SNVS\_Type \*base)  
*Get the SNVS HP security violation status flags.*
- static void `SNVS_HP_ClearSecurityViolationStatusFlags` (SNVS\_Type \*base, uint32\_t mask)  
*Clear the SNVS HP security violation status flags.*

## Driver version

- #define `FSL_SNVS_HP_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 1)`)  
Version 2.3.1.

## Initialization and deinitialization

- void `SNVS_HP_Init` (`SNVS_Type *base`)  
*Initialize the SNVS.*
- void `SNVS_HP_Deinit` (`SNVS_Type *base`)  
*Deinitialize the SNVS.*
- void `SNVS_HP_RTC_Init` (`SNVS_Type *base`, const `snvs_hp_rtc_config_t *config`)  
*Ungates the SNVS clock and configures the peripheral for basic operation.*
- void `SNVS_HP_RTC_Deinit` (`SNVS_Type *base`)  
*Stops the RTC and SRTC timers.*
- void `SNVS_HP_RTC_GetDefaultConfig` (`snvs_hp_rtc_config_t *config`)  
*Fills in the SNVS config struct with the default settings.*

## Non secure RTC current Time & Alarm

- `status_t SNVS_HP_RTC_SetDatetime` (`SNVS_Type *base`, const `snvs_hp_rtc_datetime_t *datetime`)  
*Sets the SNVS RTC date and time according to the given time structure.*
- void `SNVS_HP_RTC_GetDatetime` (`SNVS_Type *base`, `snvs_hp_rtc_datetime_t *datetime`)  
*Gets the SNVS RTC time and stores it in the given time structure.*
- `status_t SNVS_HP_RTC_SetAlarm` (`SNVS_Type *base`, const `snvs_hp_rtc_datetime_t *alarm-Time`)  
*Sets the SNVS RTC alarm time.*
- void `SNVS_HP_RTC_GetAlarm` (`SNVS_Type *base`, `snvs_hp_rtc_datetime_t *datetime`)  
*Returns the SNVS RTC alarm time.*
- void `SNVS_HP_RTC_TimeSynchronize` (`SNVS_Type *base`)  
*The function synchronizes RTC counter value with SRTC.*

## Interrupt Interface

- static void `SNVS_HP_RTC_EnableInterrupts` (`SNVS_Type *base`, `uint32_t mask`)  
*Enables the selected SNVS interrupts.*
- static void `SNVS_HP_RTC_DisableInterrupts` (`SNVS_Type *base`, `uint32_t mask`)  
*Disables the selected SNVS interrupts.*
- `uint32_t SNVS_HP_RTC_GetEnabledInterrupts` (`SNVS_Type *base`)  
*Gets the enabled SNVS interrupts.*

## Status Interface

- `uint32_t SNVS_HP_RTC_GetStatusFlags` (`SNVS_Type *base`)

*Gets the SNVS status flags.*

- static void [SNVS\\_HP\\_RTC\\_ClearStatusFlags](#) (SNVS\_Type \*base, uint32\_t mask)  
*Clears the SNVS status flags.*

## Timer Start and Stop

- static void [SNVS\\_HP\\_RTC\\_StartTimer](#) (SNVS\_Type \*base)  
*Starts the SNVS RTC time counter.*
- static void [SNVS\\_HP\\_RTC\\_StopTimer](#) (SNVS\_Type \*base)  
*Stops the SNVS RTC time counter.*

## High Assurance Counter (HAC)

- static void [SNVS\\_HP\\_EnableHighAssuranceCounter](#) (SNVS\_Type \*base, bool enable)  
*Enable or disable the High Assurance Counter (HAC)*
- static void [SNVS\\_HP\\_StartHighAssuranceCounter](#) (SNVS\_Type \*base, bool start)  
*Start or stop the High Assurance Counter (HAC)*
- static void [SNVS\\_HP\\_SetHighAssuranceCounterInitialValue](#) (SNVS\_Type \*base, uint32\_t value)  
*Set the High Assurance Counter (HAC) initialize value.*
- static void [SNVS\\_HP\\_LoadHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Load the High Assurance Counter (HAC)*
- static uint32\_t [SNVS\\_HP\\_GetHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Get the current High Assurance Counter (HAC) value.*
- static void [SNVS\\_HP\\_ClearHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Clear the High Assurance Counter (HAC)*
- static void [SNVS\\_HP\\_LockHighAssuranceCounter](#) (SNVS\_Type \*base)  
*Lock the High Assurance Counter (HAC)*

## 61.2.2 Data Structure Documentation

### 61.2.2.1 struct snvs\_hp\_rtc\_datetime\_t

#### Data Fields

- uint16\_t [year](#)  
*Range from 1970 to 2099.*
- uint8\_t [month](#)  
*Range from 1 to 12.*
- uint8\_t [day](#)  
*Range from 1 to 31 (depending on month).*
- uint8\_t [hour](#)  
*Range from 0 to 23.*
- uint8\_t [minute](#)  
*Range from 0 to 59.*
- uint8\_t [second](#)  
*Range from 0 to 59.*



## Field Documentation

- (1) `uint16_t snvs_hp_rtc_datetime_t::year`
- (2) `uint8_t snvs_hp_rtc_datetime_t::month`
- (3) `uint8_t snvs_hp_rtc_datetime_t::day`
- (4) `uint8_t snvs_hp_rtc_datetime_t::hour`
- (5) `uint8_t snvs_hp_rtc_datetime_t::minute`
- (6) `uint8_t snvs_hp_rtc_datetime_t::second`

### 61.2.2.2 struct snvs\_hp\_rtc\_config\_t

This structure holds the configuration settings for the SNVS peripheral. To initialize this structure to reasonable defaults, call the `SNVS_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

## Data Fields

- bool `rtcCalEnable`  
*true: RTC calibration mechanism is enabled; false: No calibration is used*
- `uint32_t rtcCalValue`  
*Defines signed calibration value for nonsecure RTC; This is a 5-bit 2's complement value, range from -16 to +15.*
- `uint32_t periodicInterruptFreq`  
*Defines frequency of the periodic interrupt; Range from 0 to 15.*

## 61.2.3 Macro Definition Documentation

### 61.2.3.1 #define SNVS\_MAKE\_HP\_SV\_FLAG( x ) (1U << (SNVS\_HPSVSR\_SV0\_SHIFT + (x)))

Macro help to make security violation flag `kSNVS_Violation0Flag` to `kSNVS_Violation5Flag`, For example, `SNVS_MAKE_HP_SV_FLAG(0)` is `kSNVS_Violation0Flag`.

## 61.2.4 Enumeration Type Documentation

**61.2.4.1 enum snvs\_hp\_interrupts\_t**

Enumerator

*kSNVS\_RTC\_AlarmInterrupt* RTC time alarm.  
*kSNVS\_RTC\_PeriodicInterrupt* RTC periodic interrupt.

**61.2.4.2 enum snvs\_hp\_status\_flags\_t**

Enumerator

*kSNVS\_RTC\_AlarmInterruptFlag* RTC time alarm flag.  
*kSNVS\_RTC\_PeriodicInterruptFlag* RTC periodic interrupt flag.  
*kSNVS\_ZMK\_ZeroFlag* The ZMK is zero.  
*kSNVS\_OTPMK\_ZeroFlag* The OTPMK is zero.

**61.2.4.3 enum snvs\_hp\_sv\_status\_flags\_t**

Enumerator

*kSNVS\_LP\_ViolationFlag* Low Power section Security Violation.  
*kSNVS\_ZMK\_EccFailFlag* Zeroizable Master Key Error Correcting Code Check Failure.  
*kSNVS\_LP\_SoftwareViolationFlag* LP Software Security Violation.  
*kSNVS\_FatalSoftwareViolationFlag* Software Fatal Security Violation.  
*kSNVS\_SoftwareViolationFlag* Software Security Violation.  
*kSNVS\_Violation0Flag* Security Violation 0.  
*kSNVS\_Violation1Flag* Security Violation 1.  
*kSNVS\_Violation2Flag* Security Violation 2.  
*kSNVS\_Violation4Flag* Security Violation 4.  
*kSNVS\_Violation5Flag* Security Violation 5.

**61.2.4.4 enum snvs\_hp\_ssm\_state\_t**

Enumerator

*kSNVS\_SSMInit* Init.  
*kSNVS\_SSMHardFail* Hard Fail.  
*kSNVS\_SSMSoftFail* Soft Fail.  
*kSNVS\_SSMInitInter* Init Intermediate (transition state between Init and Check)  
*kSNVS\_SSMCheck* Check.  
*kSNVS\_SSMNonSecure* Non-Secure.  
*kSNVS\_SSMTrusted* Trusted.  
*kSNVS\_SSMSecure* Secure.

## 61.2.5 Function Documentation

### 61.2.5.1 void SNVS\_HP\_Init ( SNVS\_Type \* *base* )

Note

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

Parameters

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

### 61.2.5.2 void SNVS\_HP\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

### 61.2.5.3 void SNVS\_HP\_RTC\_Init ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_config\_t \* *config* )

Note

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

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | SNVS peripheral base address                        |
| <i>config</i> | Pointer to the user's SNVS configuration structure. |

### 61.2.5.4 void SNVS\_HP\_RTC\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

### 61.2.5.5 void SNVS\_HP\_RTC\_GetDefaultConfig ( snvs\_hp\_rtc\_config\_t \* *config* )

The default values are as follows.

```

* config->rtccalenable = false;
* config->rtccalvalue = 0U;
* config->PIFreq = 0U;
*

```

## Parameters

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

#### 61.2.5.6 **status\_t SNVS\_HP\_RTC\_SetDatetime ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *datetime* )**

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

## Returns

kStatus\_Success: Success in setting the time and starting the SNVS RTC  
 kStatus\_InvalidArgument: Error because the datetime format is incorrect

#### 61.2.5.7 **void SNVS\_HP\_RTC\_GetDatetime ( SNVS\_Type \* *base*, snvs\_hp\_rtc\_datetime\_t \* *datetime* )**

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

#### 61.2.5.8 **status\_t SNVS\_HP\_RTC\_SetAlarm ( SNVS\_Type \* *base*, const snvs\_hp\_rtc\_datetime\_t \* *alarmTime* )**

The function sets the RTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

## Parameters

|                  |                                                          |
|------------------|----------------------------------------------------------|
| <i>base</i>      | SNVS peripheral base address                             |
| <i>alarmTime</i> | Pointer to the structure where the alarm time is stored. |

## Returns

kStatus\_Success: success in setting the SNVS RTC alarm  
 kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
 kStatus\_Fail: Error because the alarm time has already passed

### 61.2.5.9 void SNVS\_HP\_RTC\_GetAlarm ( SNVS\_Type \* *base*, snvs\_hp\_rtc\_datetime\_t \* *datetime* )

## Parameters

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                               |
| <i>datetime</i> | Pointer to the structure where the alarm date and time details are stored. |

### 61.2.5.10 void SNVS\_HP\_RTC\_TimeSynchronize ( SNVS\_Type \* *base* )

## Parameters

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

### 61.2.5.11 static void SNVS\_HP\_RTC\_EnableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                          |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_hp_interrupts_t |

### 61.2.5.12 static void SNVS\_HP\_RTC\_DisableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                        |
| <i>mask</i> | The interrupts to disable. This is a logical OR of members of the enumeration :: <code>_snvs_hp_interrupts_t</code> |

**61.2.5.13 `uint32_t SNVS_HP_RTC_GetEnabledInterrupts ( SNVS_Type * base )`**

## Parameters

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

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: `_snvs_hp_interrupts_t`

**61.2.5.14 `uint32_t SNVS_HP_RTC_GetStatusFlags ( SNVS_Type * base )`**

## Parameters

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

## Returns

The status flags. This is the logical OR of members of the enumeration :: `_snvs_hp_status_flags_t`

**61.2.5.15 `static void SNVS_HP_RTC_ClearStatusFlags ( SNVS_Type * base, uint32_t mask ) [inline], [static]`**

## Parameters

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                          |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration :: <code>_snvs_hp_status_flags_t</code> |

**61.2.5.16 `static void SNVS_HP_RTC_StartTimer ( SNVS_Type * base ) [inline], [static]`**

Parameters

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

**61.2.5.17 static void SNVS\_HP\_RTC\_StopTimer ( SNVS\_Type \* *base* ) [inline], [static]**

Parameters

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

**61.2.5.18 static void SNVS\_HP\_EnableMasterKeySelection ( SNVS\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

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

**61.2.5.19 static void SNVS\_HP\_ProgramZeroizableMasterKey ( SNVS\_Type \* *base* ) [inline], [static]**

Parameters

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

**61.2.5.20 static void SNVS\_HP\_ChangeSSMState ( SNVS\_Type \* *base* ) [inline], [static]**

Trigger state transition of the system security monitor (SSM). It results only the following transitions of the SSM:

- Check State -> Non-Secure (when Non-Secure Boot and not in Fab Configuration)
- Check State -> Trusted (when Secure Boot or in Fab Configuration )
- Trusted State -> Secure
- Secure State -> Trusted
- Soft Fail -> Non-Secure

Parameters

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

**61.2.5.21 static void SNVS\_HP\_SetSoftwareFatalSecurityViolation ( SNVS\_Type \* *base* )**  
**[inline], [static]**

The result SSM state transition is:

- Check State -> Soft Fail
- Non-Secure State -> Soft Fail
- Trusted State -> Soft Fail
- Secure State -> Soft Fail

Parameters

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

**61.2.5.22 static void SNVS\_HP\_SetSoftwareSecurityViolation ( SNVS\_Type \* *base* )**  
**[inline], [static]**

The result SSM state transition is:

- Check -> Non-Secure
- Trusted -> Soft Fail
- Secure -> Soft Fail

Parameters

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

**61.2.5.23 static snvs\_hp\_ssm\_state\_t SNVS\_HP\_GetSSMState ( SNVS\_Type \* *base* )**  
**[inline], [static]**

Parameters

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

Returns

Current SSM state



**61.2.5.24** `static void SNVS_HP_ResetLP ( SNVS_Type * base ) [inline], [static]`

Reset the LP section except SRTC and Time alarm.

Parameters

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

**61.2.5.25** `static void SNVS_HP_EnableHighAssuranceCounter ( SNVS_Type * base, bool enable ) [inline], [static]`

Parameters

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

**61.2.5.26** `static void SNVS_HP_StartHighAssuranceCounter ( SNVS_Type * base, bool start ) [inline], [static]`

Parameters

|              |                                    |
|--------------|------------------------------------|
| <i>base</i>  | SNVS peripheral base address       |
| <i>start</i> | Pass true to start, false to stop. |

**61.2.5.27** `static void SNVS_HP_SetHighAssuranceCounterInitialValue ( SNVS_Type * base, uint32_t value ) [inline], [static]`

Parameters

|              |                              |
|--------------|------------------------------|
| <i>base</i>  | SNVS peripheral base address |
| <i>value</i> | The initial value to set.    |

**61.2.5.28** `static void SNVS_HP_LoadHighAssuranceCounter ( SNVS_Type * base ) [inline], [static]`

This function loads the HAC initialize value to counter register.

Parameters

---

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

**61.2.5.29** `static uint32_t SNVS_HP_GetHighAssuranceCounter ( SNVS_Type * base )  
[inline], [static]`

Parameters

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

Returns

HAC current value.

**61.2.5.30** `static void SNVS_HP_ClearHighAssuranceCounter ( SNVS_Type * base )  
[inline], [static]`

This function can be called in a functional or soft fail state. When the HAC is enabled:

- If the HAC is cleared in the soft fail state, the SSM transitions to the hard fail state immediately;
- If the HAC is cleared in functional state, the SSM will transition to hard fail immediately after transitioning to soft fail.

Parameters

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

**61.2.5.31** `static void SNVS_HP_LockHighAssuranceCounter ( SNVS_Type * base )  
[inline], [static]`

Once locked, the HAC initialize value could not be changed, the HAC enable status could not be changed. This could only be unlocked by system reset.

Parameters

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

**61.2.5.32** `static uint32_t SNVS_HP_GetStatusFlags ( SNVS_Type * base ) [inline],  
[static]`

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`.

## Parameters

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

## Returns

The OR'ed value of status flags.

#### 61.2.5.33 static void SNVS\_HP\_ClearStatusFlags ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_status_flags_t`. Only these flags could be cleared using this API.

- [kSNVS\\_RTC\\_PeriodicInterruptFlag](#)
- [kSNVS\\_RTC\\_AlarmInterruptFlag](#)

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | SNVS peripheral base address       |
| <i>mask</i> | OR'ed value of the flags to clear. |

#### 61.2.5.34 static uint32\_t SNVS\_HP\_GetSecurityViolationStatusFlags ( SNVS\_Type \* *base* ) [inline], [static]

The flags are returned as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`.

## Parameters

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

## Returns

The OR'ed value of security violation status flags.

#### 61.2.5.35 static void SNVS\_HP\_ClearSecurityViolationStatusFlags ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The flags to clear are passed in as the OR'ed value of the enumeration :: `_snvs_hp_sv_status_flags_t`. Only these flags could be cleared using this API.

- [kSNVS\\_ZMK\\_EccFailFlag](#)
- [kSNVS\\_Violation0Flag](#)

- [kSNVS\\_Violation1Flag](#)
- [kSNVS\\_Violation2Flag](#)
- [kSNVS\\_Violation3Flag](#)
- [kSNVS\\_Violation4Flag](#)
- [kSNVS\\_Violation5Flag](#)

## Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | SNVS peripheral base address       |
| <i>mask</i> | OR'ed value of the flags to clear. |

## 61.3 Secure Non-Volatile Storage Low-Power

### 61.3.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Secure Non-Volatile Storage Low-Power (SNVS-LP) module.

The SNVS\_LP is a data storage subsystem. Its purpose is to store and protect system data, regardless of the main system power state. The SNVS\_LP is in the always-powered-up domain, which is a separate power domain with its own power supply.

### Data Structures

- struct [tamper\\_active\\_tx\\_config\\_t](#)  
*Structure is used to configure SNVS LP active TX tamper pins. [More...](#)*
- struct [tamper\\_active\\_rx\\_config\\_t](#)  
*Structure is used to configure SNVS LP active RX tamper pins. [More...](#)*
- struct [snvs\\_lp\\_passive\\_tamper\\_t](#)  
*Structure is used to configure SNVS LP passive tamper pins. [More...](#)*
- struct [snvs\\_lp\\_srtc\\_datetime\\_t](#)  
*Structure is used to hold the date and time. [More...](#)*
- struct [snvs\\_lp\\_srtc\\_config\\_t](#)  
*SNVS\_LP config structure. [More...](#)*

### Macros

- #define [SNVS\\_ZMK\\_REG\\_COUNT](#) 8U /\* 8 Zeroizable Master Key registers. \*/  
*Define of SNVS\_LP Zeroizable Master Key registers.*
- #define [SNVS\\_LP\\_MAX\\_TAMPER](#) kSNVS\_ExternalTamper10  
*Define of SNVS\_LP Max possible tamper.*

### Enumerations

- enum [snvs\\_lp\\_srtc\\_interrupts\\_t](#) { kSNVS\_SRTC\_AlarmInterrupt = SNVS\_LPCR\_LPTA\_EN\_MASK }
- enum [snvs\\_lp\\_srtc\\_status\\_flags\\_t](#) { kSNVS\_SRTC\_AlarmInterruptFlag = SNVS\_LPSR\_LPTA\_MASK }
- enum [snvs\\_lp\\_external\\_tamper\\_t](#)  
*List of SNVS\_LP external tamper flags.*
- enum [snvs\\_lp\\_active\\_tx\\_tamper\\_t](#)  
*List of SNVS\_LP active tamper flags.*
- enum [snvs\\_lp\\_active\\_clock\\_t](#)  
*List of SNVS\_LP active clock tamper flags.*
- enum [snvs\\_lp\\_external\\_tamper\\_status\\_t](#)

- *List of SNVS\_LP external tampers status.*
- enum [snvs\\_lp\\_external\\_tamper\\_polarity\\_t](#)  
SNVS\_LP external tamper polarity.
- enum [snvs\\_lp\\_zmk\\_program\\_mode\\_t](#) {  
kSNVS\_ZMKSoftwareProgram,  
kSNVS\_ZMKHardwareProgram }  
SNVS\_LP Zeroizable Master Key programming mode.
- enum [snvs\\_lp\\_master\\_key\\_mode\\_t](#) {  
kSNVS\_OTPMK = 0,  
kSNVS\_ZMK = 2,  
kSNVS\_CMK = 3 }  
SNVS\_LP Master Key mode.

## Functions

- void [SNVS\\_LP\\_SRTC\\_Init](#) (SNVS\_Type \*base, const [snvs\\_lp\\_srtc\\_config\\_t](#) \*config)  
Ungates the SNVS clock and configures the peripheral for basic operation.
- void [SNVS\\_LP\\_SRTC\\_Deinit](#) (SNVS\_Type \*base)  
Stops the SRTC timer.
- void [SNVS\\_LP\\_SRTC\\_GetDefaultConfig](#) ([snvs\\_lp\\_srtc\\_config\\_t](#) \*config)  
Fills in the SNVS\_LP config struct with the default settings.

## Driver version

- #define [FSL\\_SNVS\\_LP\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 4, 3))  
Version 2.4.3.

## Initialization and deinitialization

- void [SNVS\\_LP\\_Init](#) (SNVS\_Type \*base)  
Ungates the SNVS clock and configures the peripheral for basic operation.
- void [SNVS\\_LP\\_Deinit](#) (SNVS\_Type \*base)  
Deinit the SNVS LP section.

## Secure RTC (SRTC) current Time & Alarm

- [status\\_t SNVS\\_LP\\_SRTC\\_SetDatetime](#) (SNVS\_Type \*base, const [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*datetime)  
Sets the SNVS SRTC date and time according to the given time structure.
- void [SNVS\\_LP\\_SRTC\\_GetDatetime](#) (SNVS\_Type \*base, [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*datetime)  
Gets the SNVS SRTC time and stores it in the given time structure.
- [status\\_t SNVS\\_LP\\_SRTC\\_SetAlarm](#) (SNVS\_Type \*base, const [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*alarm-Time)  
Sets the SNVS SRTC alarm time.
- void [SNVS\\_LP\\_SRTC\\_GetAlarm](#) (SNVS\_Type \*base, [snvs\\_lp\\_srtc\\_datetime\\_t](#) \*datetime)

*Returns the SNVS SRTC alarm time.*

## Interrupt Interface

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

## Status Interface

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

## Timer Start and Stop

- static void [SNVS\\_LP\\_SRTC\\_StartTimer](#) (SNVS\_Type \*base)  
*Starts the SNVS SRTC time counter.*
- static void [SNVS\\_LP\\_SRTC\\_StopTimer](#) (SNVS\_Type \*base)  
*Stops the SNVS SRTC time counter.*

## External tampering

- void [SNVS\\_LP\\_EnablePassiveTamper](#) (SNVS\_Type \*base, [snvs\\_lp\\_external\\_tamper\\_t](#) pin, [snvs\\_lp\\_passive\\_tamper\\_t](#) config)  
*Enables the specified SNVS external tamper.*
- [status\\_t](#) [SNVS\\_LP\\_EnableTxActiveTamper](#) (SNVS\_Type \*base, [snvs\\_lp\\_active\\_tx\\_tamper\\_t](#) pin, [tamper\\_active\\_tx\\_config\\_t](#) config)  
*Enable active tamper tx external pad.*
- [status\\_t](#) [SNVS\\_LP\\_EnableRxActiveTamper](#) (SNVS\_Type \*base, [snvs\\_lp\\_external\\_tamper\\_t](#) rx, [tamper\\_active\\_rx\\_config\\_t](#) config)  
*Enable active tamper rx external pad.*
- [status\\_t](#) [SNVS\\_LP\\_SetVoltageTamper](#) (SNVS\_Type \*base, bool enable)  
*Sets voltage tamper detect.*
- [status\\_t](#) [SNVS\\_LP\\_SetTemperatureTamper](#) (SNVS\_Type \*base, bool enable)  
*Sets temperature tamper detect.*
- [status\\_t](#) [SNVS\\_LP\\_SetClockTamper](#) (SNVS\_Type \*base, bool enable)  
*Sets clock tamper detect.*
- [snvs\\_lp\\_external\\_tamper\\_status\\_t](#) [SNVS\\_LP\\_CheckVoltageTamper](#) (SNVS\_Type \*base)  
*brief Check voltage tamper*
- [snvs\\_lp\\_external\\_tamper\\_status\\_t](#) [SNVS\\_LP\\_CheckTemperatureTamper](#) (SNVS\_Type \*base)



- *Check temperature tamper.*
- `snvs_lp_external_tamper_status_t SNVS_LP_CheckClockTamper (SNVS_Type *base)`  
*brief Check clock tamper*
- `void SNVS_LP_TamperPinTx_GetDefaultConfig (tamper_active_tx_config_t *config)`  
*Fills in the SNVS tamper pin config struct with the default settings.*
- `void SNVS_LP_TamperPinRx_GetDefaultConfig (tamper_active_rx_config_t *config)`  
*brief Fills in the SNVS tamper pin config struct with the default settings.*
- `void SNVS_LP_PassiveTamperPin_GetDefaultConfig (snvs_lp_passive_tamper_t *config)`  
*Fills in the SNVS tamper pin config struct with the default settings.*
- `void SNVS_LP_DisableExternalTamper (SNVS_Type *base, snvs_lp_external_tamper_t pin)`  
*Disables the specified SNVS external tamper.*
- `void SNVS_LP_DisableAllExternalTamper (SNVS_Type *base)`  
*Disable all external tamper.*
- `snvs_lp_external_tamper_status_t SNVS_LP_GetExternalTamperStatus (SNVS_Type *base, snvs_lp_external_tamper_t pin)`  
*Returns status of the specified external tamper.*
- `void SNVS_LP_ClearExternalTamperStatus (SNVS_Type *base, snvs_lp_external_tamper_t pin)`  
*Clears status of the specified external tamper.*
- `void SNVS_LP_ClearAllExternalTamperStatus (SNVS_Type *base)`  
*Clears status of the all external tamper.*

## Monotonic Counter (MC)

- `static void SNVS_LP_EnableMonotonicCounter (SNVS_Type *base, bool enable)`  
*Enable or disable the Monotonic Counter.*
- `uint64_t SNVS_LP_GetMonotonicCounter (SNVS_Type *base)`  
*Get the current Monotonic Counter.*
- `static void SNVS_LP_IncreaseMonotonicCounter (SNVS_Type *base)`  
*Increase the Monotonic Counter.*

## Zeroizable Master Key (ZMK)

- `void SNVS_LP_WriteZeroizableMasterKey (SNVS_Type *base, uint32_t ZMKey[SNVS_ZMK_REG_COUNT])`  
*Write Zeroizable Master Key (ZMK) to the SNVS registers.*
- `static void SNVS_LP_SetZeroizableMasterKeyValid (SNVS_Type *base, bool valid)`  
*Set Zeroizable Master Key valid.*
- `static bool SNVS_LP_GetZeroizableMasterKeyValid (SNVS_Type *base)`  
*Get Zeroizable Master Key valid status.*
- `static void SNVS_LP_SetZeroizableMasterKeyProgramMode (SNVS_Type *base, snvs_lp_zmk_program_mode_t mode)`  
*Set Zeroizable Master Key programming mode.*
- `static void SNVS_LP_EnableZeroizableMasterKeyECC (SNVS_Type *base, bool enable)`  
*Enable or disable Zeroizable Master Key ECC.*
- `static void SNVS_LP_SetMasterKeyMode (SNVS_Type *base, snvs_lp_master_key_mode_t mode)`  
*Set SNVS Master Key mode.*

## 61.3.2 Data Structure Documentation

### 61.3.2.1 struct `tamper_active_tx_config_t`

### 61.3.2.2 struct `tamper_active_rx_config_t`

### 61.3.2.3 struct `snvs_lp_passive_tamper_t`

### 61.3.2.4 struct `snvs_lp_srtc_datetime_t`

#### Data Fields

- `uint16_t year`  
*Range from 1970 to 2099.*
- `uint8_t month`  
*Range from 1 to 12.*
- `uint8_t day`  
*Range from 1 to 31 (depending on month).*
- `uint8_t hour`  
*Range from 0 to 23.*
- `uint8_t minute`  
*Range from 0 to 59.*
- `uint8_t second`  
*Range from 0 to 59.*

#### Field Documentation

(1) `uint16_t snvs_lp_srtc_datetime_t::year`

(2) `uint8_t snvs_lp_srtc_datetime_t::month`

(3) `uint8_t snvs_lp_srtc_datetime_t::day`

(4) `uint8_t snvs_lp_srtc_datetime_t::hour`

(5) `uint8_t snvs_lp_srtc_datetime_t::minute`

(6) `uint8_t snvs_lp_srtc_datetime_t::second`

### 61.3.2.5 struct `snvs_lp_srtc_config_t`

This structure holds the configuration settings for the SNVS\_LP peripheral. To initialize this structure to reasonable defaults, call the `SNVS_LP_GetDefaultConfig()` function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

## Data Fields

- bool [srtcCalEnable](#)  
*true: SRTC calibration mechanism is enabled; false: No calibration is used*
- uint32\_t [srtcCalValue](#)  
*Defines signed calibration value for SRTC; This is a 5-bit 2's complement value, range from -16 to +15.*

## 61.3.3 Enumeration Type Documentation

### 61.3.3.1 enum snvs\_lp\_srtc\_interrupts\_t

Enumerator

***kSNVS\_SRTC\_AlarmInterrupt*** SRTC time alarm.

### 61.3.3.2 enum snvs\_lp\_srtc\_status\_flags\_t

Enumerator

***kSNVS\_SRTC\_AlarmInterruptFlag*** SRTC time alarm flag.

### 61.3.3.3 enum snvs\_lp\_zmk\_program\_mode\_t

Enumerator

***kSNVS\_ZMKSoftwareProgram*** Software programming mode.

***kSNVS\_ZMKHardwareProgram*** Hardware programming mode.

### 61.3.3.4 enum snvs\_lp\_master\_key\_mode\_t

Enumerator

***kSNVS\_OTPMK*** One Time Programmable Master Key.

***kSNVS\_ZMK*** Zeroizable Master Key.

***kSNVS\_CMK*** Combined Master Key, it is XOR of OPTMK and ZMK.

## 61.3.4 Function Documentation

### 61.3.4.1 void SNVS\_LP\_Init ( SNVS\_Type \* *base* )

Note

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

Parameters

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

#### 61.3.4.2 void SNVS\_LP\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

#### 61.3.4.3 void SNVS\_LP\_SRTC\_Init ( SNVS\_Type \* *base*, const snvs\_lp\_srtc\_config\_t \* *config* )

Note

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

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | SNVS peripheral base address                        |
| <i>config</i> | Pointer to the user's SNVS configuration structure. |

#### 61.3.4.4 void SNVS\_LP\_SRTC\_Deinit ( SNVS\_Type \* *base* )

Parameters

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

#### 61.3.4.5 void SNVS\_LP\_SRTC\_GetDefaultConfig ( snvs\_lp\_srtc\_config\_t \* *config* )

The default values are as follows.

```
* config->srtccalenable = false;
* config->srtccalvalue = 0U;
*
```

## Parameters

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

#### 61.3.4.6 **status\_t SNVS\_LP\_SRTC\_SetDatetime ( SNVS\_Type \* *base*, const snvs\_lp\_srtc\_datetime\_t \* *datetime* )**

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

## Returns

kStatus\_Success: Success in setting the time and starting the SNVS SRTC  
 kStatus\_InvalidArgument: Error because the datetime format is incorrect

#### 61.3.4.7 **void SNVS\_LP\_SRTC\_GetDatetime ( SNVS\_Type \* *base*, snvs\_lp\_srtc\_datetime\_t \* *datetime* )**

## Parameters

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                         |
| <i>datetime</i> | Pointer to the structure where the date and time details are stored. |

#### 61.3.4.8 **status\_t SNVS\_LP\_SRTC\_SetAlarm ( SNVS\_Type \* *base*, const snvs\_lp\_srtc\_datetime\_t \* *alarmTime* )**

The function sets the SRTC alarm. It also checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error. Please note, that SRTC alarm has limited resolution because only 32 most significant bits of SRTC counter are compared to SRTC Alarm register. If the alarm time is beyond SRTC resolution, the function does not set the alarm and returns an error.

## Parameters

---

|                  |                                                          |
|------------------|----------------------------------------------------------|
| <i>base</i>      | SNVS peripheral base address                             |
| <i>alarmTime</i> | Pointer to the structure where the alarm time is stored. |

## Returns

kStatus\_Success: success in setting the SNVS SRTC alarm  
 kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
 kStatus\_Fail: Error because the alarm time has already passed or is beyond resolution

#### 61.3.4.9 void SNVS\_LP\_SRTC\_GetAlarm ( SNVS\_Type \* *base*, snvs\_lp\_srtc\_datetime\_t \* *datetime* )

## Parameters

|                 |                                                                            |
|-----------------|----------------------------------------------------------------------------|
| <i>base</i>     | SNVS peripheral base address                                               |
| <i>datetime</i> | Pointer to the structure where the alarm date and time details are stored. |

#### 61.3.4.10 static void SNVS\_LP\_SRTC\_EnableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                             |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts |

#### 61.3.4.11 static void SNVS\_LP\_SRTC\_DisableInterrupts ( SNVS\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                             |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration :: _snvs_lp_srtc_interrupts |

#### 61.3.4.12 uint32\_t SNVS\_LP\_SRTC\_GetEnabledInterrupts ( SNVS\_Type \* *base* )

## Parameters

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

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration :: `_snvs_lp_srtc_interrupts`

#### 61.3.4.13 `uint32_t SNVS_LP_SRTC_GetStatusFlags ( SNVS_Type * base )`

## Parameters

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

## Returns

The status flags. This is the logical OR of members of the enumeration :: `_snvs_lp_srtc_status_flags`

#### 61.3.4.14 `static void SNVS_LP_SRTC_ClearStatusFlags ( SNVS_Type * base, uint32_t mask ) [inline], [static]`

## Parameters

|             |                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SNVS peripheral base address                                                                                             |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration :: <code>_snvs_lp_srtc_status_flags</code> |

#### 61.3.4.15 `static void SNVS_LP_SRTC_StartTimer ( SNVS_Type * base ) [inline], [static]`

## Parameters

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

#### 61.3.4.16 `static void SNVS_LP_SRTC_StopTimer ( SNVS_Type * base ) [inline], [static]`

Parameters

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

**61.3.4.17 void SNVS\_LP\_EnablePassiveTamper ( SNVS\_Type \* *base*,  
snvs\_lp\_external\_tamper\_t *pin*, snvs\_lp\_passive\_tamper\_t *config* )**

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | SNVS peripheral base address                       |
| <i>pin</i>    | SNVS external tamper pin                           |
| <i>config</i> | Configuration structure of external passive tamper |

**61.3.4.18 status\_t SNVS\_LP\_EnableTxActiveTamper ( SNVS\_Type \* *base*,  
snvs\_lp\_active\_tx\_tamper\_t *pin*, tamper\_active\_tx\_config\_t *config* )**

Parameters

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>base</i>   | SNVS peripheral base address                      |
| <i>pin</i>    | SNVS active tamper pin                            |
| <i>config</i> | Configuration structure of external active tamper |

**61.3.4.19 status\_t SNVS\_LP\_EnableRxActiveTamper ( SNVS\_Type \* *base*,  
snvs\_lp\_external\_tamper\_t *rx*, tamper\_active\_rx\_config\_t *config* )**

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | SNVS peripheral base address    |
| <i>rx</i>     | SNVS external RX tamper pin     |
| <i>config</i> | SNVS RX tamper config structure |

**61.3.4.20 status\_t SNVS\_LP\_SetVoltageTamper ( SNVS\_Type \* *base*, bool *enable* )**



Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | SNVS peripheral base address    |
| <i>enable</i> | True if enable false if disable |

**61.3.4.21** `status_t SNVS_LP_SetTemperatureTamper ( SNVS_Type * base, bool enable )`

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | SNVS peripheral base address    |
| <i>enable</i> | True if enable false if disable |

**61.3.4.22** `status_t SNVS_LP_SetClockTamper ( SNVS_Type * base, bool enable )`

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | SNVS peripheral base address    |
| <i>enable</i> | True if enable false if disable |

**61.3.4.23** `snvs_lp_external_tamper_status_t SNVS_LP_CheckVoltageTamper ( SNVS_Type * base )`

param base SNVS peripheral base address

**61.3.4.24** `snvs_lp_external_tamper_status_t SNVS_LP_CheckTemperatureTamper ( SNVS_Type * base )`

Parameters

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

**61.3.4.25** `snvs_lp_external_tamper_status_t SNVS_LP_CheckClockTamper ( SNVS_Type * base )`

param base SNVS peripheral base address

**61.3.4.26 void SNVS\_LP\_TamperPinTx\_GetDefaultConfig ( tamper\_active\_tx\_config\_t \*  
*config* )**

The default values are as follows. code config->clock = kSNVS\_ActiveTamper16HZ; config->seed = 0U; config->polynomial = 0U; encode

## Parameters

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

#### 61.3.4.27 void SNVS\_LP\_TamperPinRx\_GetDefaultConfig ( tamper\_active\_rx\_config\_t \* *config* )

The default values are as follows. code config->filterenable = 0U; config->filter = 0U; config->tx = kSNVS\_ActiveTamper1; endcode param config Pointer to the user's SNVS configuration structure.

#### 61.3.4.28 void SNVS\_LP\_PassiveTamperPin\_GetDefaultConfig ( snvs\_lp\_passive\_tamper\_t \* *config* )

The default values are as follows. code config->polarity = 0U; config->filterenable = 0U; if available on SoC config->filter = 0U; if available on SoC endcode

## Parameters

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

#### 61.3.4.29 void SNVS\_LP\_DisableExternalTamper ( SNVS\_Type \* *base*, snvs\_lp\_external\_tamper\_t *pin* )

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>pin</i>  | SNVS external tamper pin     |

#### 61.3.4.30 void SNVS\_LP\_DisableAllExternalTamper ( SNVS\_Type \* *base* )

## Parameters

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

#### 61.3.4.31 snvs\_lp\_external\_tamper\_status\_t SNVS\_LP\_GetExternalTamperStatus ( SNVS\_Type \* *base*, snvs\_lp\_external\_tamper\_t *pin* )

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>pin</i>  | SNVS external tamper pin     |

## Returns

The status flag. This is the enumeration :: `_snvs_lp_external_tamper_status`

#### 61.3.4.32 void SNVS\_LP\_ClearExternalTamperStatus ( SNVS\_Type \* *base*, snvs\_lp\_external\_tamper\_t *pin* )

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>pin</i>  | SNVS external tamper pin     |

#### 61.3.4.33 void SNVS\_LP\_ClearAllExternalTamperStatus ( SNVS\_Type \* *base* )

## Parameters

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

#### 61.3.4.34 static void SNVS\_LP\_EnableMonotonicCounter ( SNVS\_Type \* *base*, bool *enable* ) [inline], [static]

## Parameters

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

#### 61.3.4.35 uint64\_t SNVS\_LP\_GetMonotonicCounter ( SNVS\_Type \* *base* )

## Parameters

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

## Returns

Current Monotonic Counter value.

**61.3.4.36 static void SNVS\_LP\_IncreaseMonotonicCounter ( SNVS\_Type \* *base* )  
[inline], [static]**

Increase the Monotonic Counter by 1.

## Parameters

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

**61.3.4.37 void SNVS\_LP\_WriteZeroizableMasterKey ( SNVS\_Type \* *base*, uint32\_t  
*ZMKey*[SNVS\_ZMK\_REG\_COUNT] )**

## Parameters

|              |                                     |
|--------------|-------------------------------------|
| <i>base</i>  | SNVS peripheral base address        |
| <i>ZMKey</i> | The ZMK write to the SNVS register. |

**61.3.4.38 static void SNVS\_LP\_SetZeroizableMasterKeyValid ( SNVS\_Type \* *base*, bool  
*valid* ) [inline], [static]**

This API could only be called when using software programming mode. After writing ZMK using [SNV-S\\_LP\\_WriteZeroizableMasterKey](#), call this API to make the ZMK valid.

## Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>base</i>  | SNVS peripheral base address                  |
| <i>valid</i> | Pass true to set valid, false to set invalid. |

**61.3.4.39 static bool SNVS\_LP\_GetZeroizableMasterKeyValid ( SNVS\_Type \* *base* )  
[inline], [static]**

In hardware programming mode, call this API to check whether the ZMK is valid.

## Parameters

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

## Returns

true if valid, false if invalid.

**61.3.4.40** `static void SNVS_LP_SetZeroizableMasterKeyProgramMode ( SNVS_Type * base, snvs_lp_zmk_program_mode_t mode ) [inline], [static]`

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>mode</i> | ZMK programming mode.        |

**61.3.4.41** `static void SNVS_LP_EnableZeroizableMasterKeyECC ( SNVS_Type * base, bool enable ) [inline], [static]`

## Parameters

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

**61.3.4.42** `static void SNVS_LP_SetMasterKeyMode ( SNVS_Type * base, snvs_lp_master_key_mode_t mode ) [inline], [static]`

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | SNVS peripheral base address |
| <i>mode</i> | Master Key mode.             |

## Note

When [kSNVS\\_ZMK](#) or [kSNVS\\_CMK](#) used, the SNVS\_HP must be configured to enable the master key selection.

## Chapter 62

# SPDIF: Sony/Philips Digital Interface

### 62.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Sony/Philips Digital Interface (SPDIF) module of MCUXpresso SDK devices.

SPDIF driver includes functional APIs and transactional APIs.

Functional APIs target low-level APIs. Functional APIs can be used for SPDIF initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPDIF peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPDIF 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 `spdif_handle_t` as the first parameter. Initialize the handle by calling the [SPDIF\\_TransferTxCreateHandle\(\)](#) or [SPDIF\\_TransferRxCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SPDIF\\_TransferSendNonBlocking\(\)](#) and [SPDIF\\_TransferReceiveNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SPDIF_TxIdle` and `kStatus_SPDIF_RxIdle` status.

### 62.2 Typical use case

#### 62.2.1 SPDIF Send/receive using an interrupt method

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

#### 62.2.2 SPDIF Send/receive using a DMA method

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

### Modules

- [SPDIF eDMA Driver](#)

### Data Structures

- struct [spdif\\_config\\_t](#)

- *SPDIF user configuration structure. [More...](#)*
- struct `spdif_transfer_t`  
*SPDIF transfer structure. [More...](#)*
- struct `spdif_handle_t`  
*SPDIF handle structure. [More...](#)*

## Macros

- #define `SPDIF_XFER_QUEUE_SIZE` (4U)  
*SPDIF transfer queue size, user can refine it according to use case.*

## Typedefs

- typedef void(\* `spdif_transfer_callback_t` )(SPDIF\_Type \*base, spdif\_handle\_t \*handle, `status_t` status, void \*userData)  
*SPDIF transfer callback prototype.*

## Enumerations

- enum {  
`kStatus_SPDIF_RxDPLLLocked` = MAKE\_STATUS(kStatusGroup\_SPDIF, 0),  
`kStatus_SPDIF_TxFIFOError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 1),  
`kStatus_SPDIF_TxFIFOResync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 2),  
`kStatus_SPDIF_RxCnew` = MAKE\_STATUS(kStatusGroup\_SPDIF, 3),  
`kStatus_SPDIF_ValidatyNoGood` = MAKE\_STATUS(kStatusGroup\_SPDIF, 4),  
`kStatus_SPDIF_RxIllegalSymbol` = MAKE\_STATUS(kStatusGroup\_SPDIF, 5),  
`kStatus_SPDIF_RxParityBitError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 6),  
`kStatus_SPDIF_UChannelOverrun` = MAKE\_STATUS(kStatusGroup\_SPDIF, 7),  
`kStatus_SPDIF_QChannelOverrun` = MAKE\_STATUS(kStatusGroup\_SPDIF, 8),  
`kStatus_SPDIF_UQChannelSync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 9),  
`kStatus_SPDIF_UQChannelFrameError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 10),  
`kStatus_SPDIF_RxFIFOError` = MAKE\_STATUS(kStatusGroup\_SPDIF, 11),  
`kStatus_SPDIF_RxFIFOResync` = MAKE\_STATUS(kStatusGroup\_SPDIF, 12),  
`kStatus_SPDIF_LockLoss` = MAKE\_STATUS(kStatusGroup\_SPDIF, 13),  
`kStatus_SPDIF_TxIdle` = MAKE\_STATUS(kStatusGroup\_SPDIF, 14),  
`kStatus_SPDIF_RxIdle` = MAKE\_STATUS(kStatusGroup\_SPDIF, 15),  
`kStatus_SPDIF_QueueFull` = MAKE\_STATUS(kStatusGroup\_SPDIF, 16) }  
*SPDIF return status.*
- enum `spdif_rxfull_select_t` {  
`kSPDIF_RxFull1Sample` = 0x0u,  
`kSPDIF_RxFull4Samples`,  
`kSPDIF_RxFull8Samples`,  
`kSPDIF_RxFull16Samples` }  
*SPDIF Rx FIFO full flag select, it decides when assert the rx full flag.*
- enum `spdif_txempty_select_t` {  
`kSPDIF_TxEmpty0Sample` = 0x0u,  
`kSPDIF_TxEmpty4Samples`,  
`kSPDIF_TxEmpty8Samples`,



`kSPDIF_TxEmpty12Samples` }

*SPDIF tx FIFO EMPTY flag select, it decides when assert the tx empty flag.*

- enum `spdif_uchannel_source_t` {  
`kSPDIF_NoUChannel` = 0x0U,  
`kSPDIF_UChannelFromRx` = 0x1U,  
`kSPDIF_UChannelFromTx` = 0x3U }

*SPDIF U channel source.*

- enum `spdif_gain_select_t` {  
`kSPDIF_GAIN_24` = 0x0U,  
`kSPDIF_GAIN_16`,  
`kSPDIF_GAIN_12`,  
`kSPDIF_GAIN_8`,  
`kSPDIF_GAIN_6`,  
`kSPDIF_GAIN_4`,  
`kSPDIF_GAIN_3` }

*SPDIF clock gain.*

- enum `spdif_tx_source_t` {  
`kSPDIF_txFromReceiver` = 0x1U,  
`kSPDIF_txNormal` = 0x5U }

*SPDIF tx data source.*

- enum `spdif_validity_config_t` {  
`kSPDIF_validityFlagAlwaysSet` = 0x0U,  
`kSPDIF_validityFlagAlwaysClear` }

*SPDIF tx data source.*

- enum {  
`kSPDIF_RxDPLLLocked` = SPDIF\_SIE\_LOCK\_MASK,  
`kSPDIF_TxFIFOError` = SPDIF\_SIE\_TXUNOV\_MASK,  
`kSPDIF_TxFIFOResync` = SPDIF\_SIE\_TXRESYN\_MASK,  
`kSPDIF_RxControlChannelChange` = SPDIF\_SIE\_CNEW\_MASK,  
`kSPDIF_VValidityFlagNoGood` = SPDIF\_SIE\_VALNOGOOD\_MASK,  
`kSPDIF_RxIllegalSymbol` = SPDIF\_SIE\_SYMERR\_MASK,  
`kSPDIF_RxParityBitError` = SPDIF\_SIE\_BITERR\_MASK,  
`kSPDIF_UChannelReceiveRegisterFull` = SPDIF\_SIE\_URXFUL\_MASK,  
`kSPDIF_UChannelReceiveRegisterOverrun` = SPDIF\_SIE\_URXOV\_MASK,  
`kSPDIF_QChannelReceiveRegisterFull` = SPDIF\_SIE\_QRXFUL\_MASK,  
`kSPDIF_QChannelReceiveRegisterOverrun` = SPDIF\_SIE\_QRXOV\_MASK,  
`kSPDIF_UQChannelSync` = SPDIF\_SIE\_UQSYNC\_MASK,  
`kSPDIF_UQChannelFrameError` = SPDIF\_SIE\_UQERR\_MASK,  
`kSPDIF_RxFIFOError` = SPDIF\_SIE\_RXFIFOUNOV\_MASK,  
`kSPDIF_RxFIFOResync` = SPDIF\_SIE\_RXFIFORESYN\_MASK,  
`kSPDIF_LockLoss` = SPDIF\_SIE\_LOCKLOSS\_MASK,  
`kSPDIF_TxFIFOEmpty` = SPDIF\_SIE\_TXEM\_MASK,  
`kSPDIF_RxFIFOFull` = SPDIF\_SIE\_RXFIFOFUL\_MASK,  
`kSPDIF_AllInterrupt` }

*The SPDIF interrupt enable flag.*

- enum {

```
kSPDIF_RxDMAEnable = SPDIF_SCR_DMA_RX_EN_MASK,
kSPDIF_TxDMAEnable = SPDIF_SCR_DMA_TX_EN_MASK }
```

*The DMA request sources.*

## Driver version

- #define `FSL_SPDIF_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 6)`)  
*Version 2.0.6.*

## Initialization and deinitialization

- void `SPDIF_Init` (`SPDIF_Type *base`, const `spdif_config_t *config`)  
*Initializes the SPDIF peripheral.*
- void `SPDIF_GetDefaultConfig` (`spdif_config_t *config`)  
*Sets the SPDIF configuration structure to default values.*
- void `SPDIF_Deinit` (`SPDIF_Type *base`)  
*De-initializes the SPDIF peripheral.*
- uint32\_t `SPDIF_GetInstance` (`SPDIF_Type *base`)  
*Get the instance number for SPDIF.*
- static void `SPDIF_TxFIFOReset` (`SPDIF_Type *base`)  
*Resets the SPDIF Tx.*
- static void `SPDIF_RxFIFOReset` (`SPDIF_Type *base`)  
*Resets the SPDIF Rx.*
- void `SPDIF_TxEnable` (`SPDIF_Type *base`, bool enable)  
*Enables/disables the SPDIF Tx.*
- static void `SPDIF_RxEnable` (`SPDIF_Type *base`, bool enable)  
*Enables/disables the SPDIF Rx.*

## Status

- static uint32\_t `SPDIF_GetStatusFlag` (`SPDIF_Type *base`)  
*Gets the SPDIF status flag state.*
- static void `SPDIF_ClearStatusFlags` (`SPDIF_Type *base`, uint32\_t mask)  
*Clears the SPDIF status flag state.*

## Interrupts

- static void `SPDIF_EnableInterrupts` (`SPDIF_Type *base`, uint32\_t mask)  
*Enables the SPDIF Tx interrupt requests.*
- static void `SPDIF_DisableInterrupts` (`SPDIF_Type *base`, uint32\_t mask)  
*Disables the SPDIF Tx interrupt requests.*

## DMA Control

- static void `SPDIF_EnableDMA` (`SPDIF_Type *base`, uint32\_t mask, bool enable)  
*Enables/disables the SPDIF DMA requests.*
- static uint32\_t `SPDIF_TxGetLeftDataRegisterAddress` (`SPDIF_Type *base`)  
*Gets the SPDIF Tx left data register address.*
- static uint32\_t `SPDIF_TxGetRightDataRegisterAddress` (`SPDIF_Type *base`)  
*Gets the SPDIF Tx right data register address.*

- static uint32\_t [SPDIF\\_RxGetLeftDataRegisterAddress](#) (SPDIF\_Type \*base)  
*Gets the SPDIF Rx left data register address.*
- static uint32\_t [SPDIF\\_RxGetRightDataRegisterAddress](#) (SPDIF\_Type \*base)  
*Gets the SPDIF Rx right data register address.*

## Bus Operations

- void [SPDIF\\_TxSetSampleRate](#) (SPDIF\_Type \*base, uint32\_t sampleRate\_Hz, uint32\_t sourceClockFreq\_Hz)  
*Configures the SPDIF Tx sample rate.*
- uint32\_t [SPDIF\\_GetRxSampleRate](#) (SPDIF\_Type \*base, uint32\_t clockSourceFreq\_Hz)  
*Configures the SPDIF Rx audio format.*
- void [SPDIF\\_WriteBlocking](#) (SPDIF\_Type \*base, uint8\_t \*buffer, uint32\_t size)  
*Sends data using a blocking method.*
- static void [SPDIF\\_WriteLeftData](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- static void [SPDIF\\_WriteRightData](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- static void [SPDIF\\_WriteChannelStatusHigh](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- static void [SPDIF\\_WriteChannelStatusLow](#) (SPDIF\_Type \*base, uint32\_t data)  
*Writes data into SPDIF FIFO.*
- void [SPDIF\\_ReadBlocking](#) (SPDIF\_Type \*base, uint8\_t \*buffer, uint32\_t size)  
*Receives data using a blocking method.*
- static uint32\_t [SPDIF\\_ReadLeftData](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadRightData](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadChannelStatusHigh](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadChannelStatusLow](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadQChannel](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*
- static uint32\_t [SPDIF\\_ReadUChannel](#) (SPDIF\_Type \*base)  
*Reads data from the SPDIF FIFO.*

## Transactional

- void [SPDIF\\_TransferTxCreateHandle](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPDIF Tx handle.*
- void [SPDIF\\_TransferRxCreateHandle](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPDIF Rx handle.*
- status\_t [SPDIF\\_TransferSendNonBlocking](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_t](#) \*xfer)  
*Performs an interrupt non-blocking send transfer on SPDIF.*
- status\_t [SPDIF\\_TransferReceiveNonBlocking](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, [spdif\\_transfer\\_t](#) \*xfer)

- *Performs an interrupt non-blocking receive transfer on SPDIF.*
- [status\\_t SPDIF\\_TransferGetSendCount](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, size\_t \*count)  
*Gets a set byte count.*
- [status\\_t SPDIF\\_TransferGetReceiveCount](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle, size\_t \*count)  
*Gets a received byte count.*
- [void SPDIF\\_TransferAbortSend](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Aborts the current send.*
- [void SPDIF\\_TransferAbortReceive](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Aborts the current IRQ receive.*
- [void SPDIF\\_TransferTxHandleIRQ](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Tx interrupt handler.*
- [void SPDIF\\_TransferRxHandleIRQ](#) (SPDIF\_Type \*base, spdif\_handle\_t \*handle)  
*Rx interrupt handler.*

## 62.3 Data Structure Documentation

### 62.3.1 struct spdif\_config\_t

#### Data Fields

- [bool isTxAutoSync](#)  
*If auto sync mechanism open.*
- [bool isRxAutoSync](#)  
*If auto sync mechanism open.*
- [uint8\\_t DPLLClkSource](#)  
*SPDIF DPLL clock source, range from 0~15, meaning is chip-specific.*
- [uint8\\_t txClkSource](#)  
*SPDIF tx clock source, range from 0~7, meaning is chip-specific.*
- [spdif\\_rxfull\\_select\\_t rxFullSelect](#)  
*SPDIF rx buffer full select.*
- [spdif\\_txempty\\_select\\_t txFullSelect](#)  
*SPDIF tx buffer empty select.*
- [spdif\\_uchannel\\_source\\_t uChannelSrc](#)  
*U channel source.*
- [spdif\\_tx\\_source\\_t txSource](#)  
*SPDIF tx data source.*
- [spdif\\_validity\\_config\\_t validityConfig](#)  
*Validity flag config.*
- [spdif\\_gain\\_select\\_t gain](#)  
*Rx receive clock measure gain parameter.*

#### Field Documentation

##### (1) [spdif\\_gain\\_select\\_t](#) [spdif\\_config\\_t::gain](#)

### 62.3.2 struct spdif\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)  
*Data start address to transfer.*
- uint8\_t \* [qdata](#)  
*Data buffer for Q channel.*
- uint8\_t \* [udata](#)  
*Data buffer for C channel.*
- size\_t [dataSize](#)  
*Transfer size.*

#### Field Documentation

(1) uint8\_t\* [spdif\\_transfer\\_t::data](#)

(2) size\_t [spdif\\_transfer\\_t::dataSize](#)

### 62.3.3 struct \_spdif\_handle

#### Data Fields

- uint32\_t [state](#)  
*Transfer status.*
- [spdif\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function called at transfer event.*
- void \* [userData](#)  
*Callback parameter passed to callback function.*
- [spdif\\_transfer\\_t](#) [spdifQueue](#) [[SPDIF\\_XFER\\_QUEUE\\_SIZE](#)]  
*Transfer queue storing queued transfer.*
- size\_t [transferSize](#) [[SPDIF\\_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.*

## 62.4 Macro Definition Documentation

### 62.4.1 #define SPDIF\_XFER\_QUEUE\_SIZE (4U)

## 62.5 Enumeration Type Documentation

### 62.5.1 anonymous enum

Enumerator

*kStatus\_SPDIF\_RxDPLLLocked* SPDIF Rx PLL locked.  
*kStatus\_SPDIF\_TxFIFOError* SPDIF Tx FIFO error.  
*kStatus\_SPDIF\_TxFIFOResync* SPDIF Tx left and right FIFO resync.  
*kStatus\_SPDIF\_RxCnew* SPDIF Rx status channel value updated.  
*kStatus\_SPDIF\_ValidatyNoGood* SPDIF validaty flag not good.  
*kStatus\_SPDIF\_RxIllegalSymbol* SPDIF Rx receive illegal symbol.  
*kStatus\_SPDIF\_RxParityBitError* SPDIF Rx parity bit error.  
*kStatus\_SPDIF\_UChannelOverrun* SPDIF receive U channel overrun.  
*kStatus\_SPDIF\_QChannelOverrun* SPDIF receive Q channel overrun.  
*kStatus\_SPDIF\_UQChannelSync* SPDIF U/Q channel sync found.  
*kStatus\_SPDIF\_UQChannelFrameError* SPDIF U/Q channel frame error.  
*kStatus\_SPDIF\_RxFIFOError* SPDIF Rx FIFO error.  
*kStatus\_SPDIF\_RxFIFOResync* SPDIF Rx left and right FIFO resync.  
*kStatus\_SPDIF\_LockLoss* SPDIF Rx PLL clock lock loss.  
*kStatus\_SPDIF\_TxIdle* SPDIF Tx is idle.  
*kStatus\_SPDIF\_RxIdle* SPDIF Rx is idle.  
*kStatus\_SPDIF\_QueueFull* SPDIF queue full.

### 62.5.2 enum spdif\_rxfull\_select\_t

Enumerator

*kSPDIF\_RxFull1Sample* Rx full at least 1 sample in left and right FIFO.  
*kSPDIF\_RxFull4Samples* Rx full at least 4 sample in left and right FIFO.  
*kSPDIF\_RxFull8Samples* Rx full at least 8 sample in left and right FIFO.  
*kSPDIF\_RxFull16Samples* Rx full at least 16 sample in left and right FIFO.

### 62.5.3 enum spdif\_txempty\_select\_t

Enumerator

*kSPDIF\_TxEmpty0Sample* Tx empty at most 0 sample in left and right FIFO.  
*kSPDIF\_TxEmpty4Samples* Tx empty at most 4 sample in left and right FIFO.  
*kSPDIF\_TxEmpty8Samples* Tx empty at most 8 sample in left and right FIFO.  
*kSPDIF\_TxEmpty12Samples* Tx empty at most 12 sample in left and right FIFO.

### 62.5.4 enum spdif\_uchannel\_source\_t

Enumerator

*kSPDIF\_NoUChannel* No embedded U channel.  
*kSPDIF\_UChannelFromRx* U channel from receiver, it is CD mode.  
*kSPDIF\_UChannelFromTx* U channel from on chip tx.

### 62.5.5 enum spdif\_gain\_select\_t

Enumerator

*kSPDIF\_GAIN\_24* Gain select is 24.  
*kSPDIF\_GAIN\_16* Gain select is 16.  
*kSPDIF\_GAIN\_12* Gain select is 12.  
*kSPDIF\_GAIN\_8* Gain select is 8.  
*kSPDIF\_GAIN\_6* Gain select is 6.  
*kSPDIF\_GAIN\_4* Gain select is 4.  
*kSPDIF\_GAIN\_3* Gain select is 3.

### 62.5.6 enum spdif\_tx\_source\_t

Enumerator

*kSPDIF\_txFromReceiver* Tx data directly through SPDIF receiver.  
*kSPDIF\_txNormal* Normal operation, data from processor.

### 62.5.7 enum spdif\_validity\_config\_t

Enumerator

*kSPDIF\_validityFlagAlwaysSet* Outgoing validity flags always set.  
*kSPDIF\_validityFlagAlwaysClear* Outgoing validity flags always clear.

### 62.5.8 anonymous enum

Enumerator

*kSPDIF\_RxDPLLLocked* SPDIF DPLL locked.  
*kSPDIF\_TxFIFOError* Tx FIFO underrun or overrun.  
*kSPDIF\_TxFIFOResync* Tx FIFO left and right channel resync.

*kSPDIF\_RxControlChannelChange* SPDIF Rx control channel value changed.  
*kSPDIF\_ValidityFlagNoGood* SPDIF validity flag no good.  
*kSPDIF\_RxIllegalSymbol* SPDIF receiver found illegal symbol.  
*kSPDIF\_RxParityBitError* SPDIF receiver found parity bit error.  
*kSPDIF\_UChannelReceiveRegisterFull* SPDIF U channel receive register full.  
*kSPDIF\_UChannelReceiveRegisterOverflow* SPDIF U channel receive register overflow.  
*kSPDIF\_QChannelReceiveRegisterFull* SPDIF Q channel receive register full.  
*kSPDIF\_QChannelReceiveRegisterOverflow* SPDIF Q channel receive register overflow.  
*kSPDIF\_UQChannelSync* SPDIF U/Q channel sync found.  
*kSPDIF\_UQChannelFrameError* SPDIF U/Q channel frame error.  
*kSPDIF\_RxFIFOError* SPDIF Rx FIFO underrun/overflow.  
*kSPDIF\_RxFIFOResync* SPDIF Rx left and right FIFO resync.  
*kSPDIF\_LockLoss* SPDIF receiver loss of lock.  
*kSPDIF\_TxFIFOEmpty* SPDIF Tx FIFO empty.  
*kSPDIF\_RxFIFOFull* SPDIF Rx FIFO full.  
*kSPDIF\_AllInterrupt* all interrupt

### 62.5.9 anonymous enum

Enumerator

*kSPDIF\_RxDMAEnable* Rx FIFO full.  
*kSPDIF\_TxDMAEnable* Tx FIFO empty.

## 62.6 Function Documentation

### 62.6.1 void SPDIF\_Init ( SPDIF\_Type \* *base*, const spdif\_config\_t \* *config* )

Un-gates the SPDIF clock, resets the module, and configures SPDIF with a configuration structure. The configuration structure can be custom filled or set with default values by [SPDIF\\_GetDefaultConfig\(\)](#).

Note

This API should be called at the beginning of the application to use the SPDIF driver. Otherwise, accessing the SPDIF module can cause a hard fault because the clock is not enabled.

Parameters

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



|               |                                |
|---------------|--------------------------------|
| <i>config</i> | SPDIF configuration structure. |
|---------------|--------------------------------|

### 62.6.2 void SPDIF\_GetDefaultConfig ( spdif\_config\_t \* *config* )

This API initializes the configuration structure for use in SPDIF\_Init. The initialized structure can remain unchanged in SPDIF\_Init, or it can be modified before calling SPDIF\_Init. This is an example.

```
spdif_config_t config;
SPDIF_GetDefaultConfig(&config);
```

Parameters

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

### 62.6.3 void SPDIF\_Deinit ( SPDIF\_Type \* *base* )

This API gates the SPDIF clock. The SPDIF module can't operate unless SPDIF\_Init is called to enable the clock.

Parameters

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

### 62.6.4 uint32\_t SPDIF\_GetInstance ( SPDIF\_Type \* *base* )

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

### 62.6.5 static void SPDIF\_TxFIFOReset ( SPDIF\_Type \* *base* ) [inline], [static]

This function makes Tx FIFO in reset mode.

Parameters

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

**62.6.6 static void SPDIF\_RxFIFOReset ( SPDIF\_Type \* *base* ) [inline], [static]**

This function enables the software reset and FIFO reset of SPDIF Rx. After reset, clear the reset bit.

Parameters

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

**62.6.7 void SPDIF\_TxEnable ( SPDIF\_Type \* *base*, bool *enable* )**

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | SPDIF base pointer                               |
| <i>enable</i> | True means enable SPDIF Tx, false means disable. |

**62.6.8 static void SPDIF\_RxEnable ( SPDIF\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | SPDIF base pointer                               |
| <i>enable</i> | True means enable SPDIF Rx, false means disable. |

**62.6.9 static uint32\_t SPDIF\_GetStatusFlag ( SPDIF\_Type \* *base* ) [inline], [static]**

Parameters

---

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

## Returns

SPDIF status flag value. Use the `_spdif_interrupt_enable_t` to get the status value needed.

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

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPDIF base pointer                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>mask</i> | State mask. It can be a combination of the <code>_spdif_interrupt_enable_t</code> member. Notice these members cannot be included, as these flags cannot be cleared by writing 1 to these bits: <ul style="list-style-type: none"> <li>• <code>kSPDIF_UChannelReceiveRegisterFull</code></li> <li>• <code>kSPDIF_QChannelReceiveRegisterFull</code></li> <li>• <code>kSPDIF_TxFIFOEmpty</code></li> <li>• <code>kSPDIF_RxFIFOFull</code></li> </ul> |

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

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPDIF 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>• <code>kSPDIF_WordStartInterruptEnable</code></li> <li>• <code>kSPDIF_SyncErrorInterruptEnable</code></li> <li>• <code>kSPDIF_FIFOWarningInterruptEnable</code></li> <li>• <code>kSPDIF_FIFORequestInterruptEnable</code></li> <li>• <code>kSPDIF_FIFOErrorInterruptEnable</code></li> </ul> |

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

## Parameters

|             |                                                                                                                                                                                                                                                                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPDIF 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>• kSPDIF_WordStartInterruptEnable</li> <li>• kSPDIF_SyncErrorInterruptEnable</li> <li>• kSPDIF_FIFOWarningInterruptEnable</li> <li>• kSPDIF_FIFORequestInterruptEnable</li> <li>• kSPDIF_FIFOErrorInterruptEnable</li> </ul> |

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

## Parameters

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

**62.6.14 static uint32\_t SPDIF\_TxGetLeftDataRegisterAddress ( SPDIF\_Type \* *base* ) [inline], [static]**

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

## Returns

data register address.

**62.6.15** `static uint32_t SPDIF_TxGetRightDataRegisterAddress ( SPDIF_Type *  
base ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

## Returns

data register address.

**62.6.16** `static uint32_t SPDIF_RxGetLeftDataRegisterAddress ( SPDIF_Type * base ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

## Returns

data register address.

**62.6.17** `static uint32_t SPDIF_RxGetRightDataRegisterAddress ( SPDIF_Type * base ) [inline], [static]`

This API is used to provide a transfer address for the SPDIF DMA transfer configuration.

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

## Returns

data register address.

**62.6.18** `void SPDIF_TxSetSampleRate ( SPDIF_Type * base, uint32_t sampleRate_Hz, uint32_t sourceClockFreq_Hz )`

The audio format can be changed at run-time. This function configures the sample rate.

## Parameters

|                            |                                        |
|----------------------------|----------------------------------------|
| <i>base</i>                | SPDIF base pointer.                    |
| <i>sampleRate_Hz</i>       | SPDIF sample rate frequency in Hz.     |
| <i>sourceClock-Freq_Hz</i> | SPDIF tx clock source frequency in Hz. |

### 62.6.19 uint32\_t SPDIF\_GetRxSampleRate ( SPDIF\_Type \* *base*, uint32\_t *clockSourceFreq\_Hz* )

The audio format can be changed at run-time. This function configures the sample rate and audio data format to be transferred.

## Parameters

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>base</i>                | SPDIF base pointer.                 |
| <i>clockSource-Freq_Hz</i> | SPDIF system clock frequency in hz. |

### 62.6.20 void SPDIF\_WriteBlocking ( SPDIF\_Type \* *base*, uint8\_t \* *buffer*, uint32\_t *size* )

## Note

This function blocks by polling until data is ready to be sent.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SPDIF base pointer.                |
| <i>buffer</i> | Pointer to the data to be written. |
| <i>size</i>   | Bytes to be written.               |

### 62.6.21 static void SPDIF\_WriteLeftData ( SPDIF\_Type \* *base*, uint32\_t *data* ) [inline], [static]

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | SPDIF base pointer.       |
| <i>data</i> | Data needs to be written. |

**62.6.22** `static void SPDIF_WriteRightData ( SPDIF_Type * base, uint32_t data )`  
**[inline], [static]**

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | SPDIF base pointer.       |
| <i>data</i> | Data needs to be written. |

**62.6.23** `static void SPDIF_WriteChannelStatusHigh ( SPDIF_Type * base, uint32_t`  
***data* ) [inline], [static]**

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | SPDIF base pointer.       |
| <i>data</i> | Data needs to be written. |

**62.6.24** `static void SPDIF_WriteChannelStatusLow ( SPDIF_Type * base, uint32_t`  
***data* ) [inline], [static]**

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | SPDIF base pointer.       |
| <i>data</i> | Data needs to be written. |

**62.6.25** `void SPDIF_ReadBlocking ( SPDIF_Type * base, uint8_t * buffer, uint32_t`  
***size* )**

## Note

This function blocks by polling until data is ready to be sent.



## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | SPDIF base pointer.             |
| <i>buffer</i> | Pointer to the data to be read. |
| <i>size</i>   | Bytes to be read.               |

**62.6.26** `static uint32_t SPDIF_ReadLeftData ( SPDIF_Type * base ) [inline],  
[static]`

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

## Returns

Data in SPDIF FIFO.

**62.6.27** `static uint32_t SPDIF_ReadRightData ( SPDIF_Type * base ) [inline],  
[static]`

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

## Returns

Data in SPDIF FIFO.

**62.6.28** `static uint32_t SPDIF_ReadChannelStatusHigh ( SPDIF_Type * base )  
[inline], [static]`

## Parameters

---

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

Returns

Data in SPDIF FIFO.

**62.6.29** `static uint32_t SPDIF_ReadChannelStatusLow ( SPDIF_Type * base )  
[inline], [static]`

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

Returns

Data in SPDIF FIFO.

**62.6.30** `static uint32_t SPDIF_ReadQChannel ( SPDIF_Type * base ) [inline],  
[static]`

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

Returns

Data in SPDIF FIFO.

**62.6.31** `static uint32_t SPDIF_ReadUChannel ( SPDIF_Type * base ) [inline],  
[static]`

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

Returns

Data in SPDIF FIFO.

**62.6.32** void SPDIF\_TransferTxCreateHandle ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, spdif\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the Tx handle for the SPDIF Tx transactional APIs. Call this function once to get the handle initialized.

## Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>base</i>     | SPDIF base pointer                             |
| <i>handle</i>   | SPDIF handle pointer.                          |
| <i>callback</i> | Pointer to the user callback function.         |
| <i>userData</i> | User parameter passed to the callback function |

### 62.6.33 void SPDIF\_TransferRxCreateHandle ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, spdif\_transfer\_callback\_t *callback*, void \* *userData* )

This function initializes the Rx handle for the SPDIF Rx transactional APIs. Call this function once to get the handle initialized.

## Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>base</i>     | SPDIF base pointer.                             |
| <i>handle</i>   | SPDIF handle pointer.                           |
| <i>callback</i> | Pointer to the user callback function.          |
| <i>userData</i> | User parameter passed to the callback function. |

### 62.6.34 status\_t SPDIF\_TransferSendNonBlocking ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, spdif\_transfer\_t \* *xfer* )

## Note

This API returns immediately after the transfer initiates. Call the SPDIF\_TxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SPDIF\_Busy, the transfer is finished.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | SPDIF base pointer.                                                      |
| <i>handle</i> | Pointer to the spdif_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">spdif_transfer_t</a> structure.               |

## Return values

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

### 62.6.35 **status\_t SPDIF\_TransferReceiveNonBlocking ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, spdif\_transfer\_t \* *xfer* )**

## Note

This API returns immediately after the transfer initiates. Call the SPDIF\_RxGetTransferStatusIRQ to poll the transfer status and check whether the transfer is finished. If the return status is not kStatus\_SPDIF\_Busy, the transfer is finished.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | SPDIF base pointer                                                       |
| <i>handle</i> | Pointer to the spdif_handle_t structure which stores the transfer state. |
| <i>xfer</i>   | Pointer to the <a href="#">spdif_transfer_t</a> structure.               |

## Return values

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

### 62.6.36 **status\_t SPDIF\_TransferGetSendCount ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, size\_t \* *count* )**

## Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SPDIF base pointer. |
|-------------|---------------------|

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>handle</i> | Pointer to the spdif_handle_t structure which stores the transfer state. |
| <i>count</i>  | Bytes count sent.                                                        |

Return values

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

### 62.6.37 status\_t SPDIF\_TransferGetReceiveCount ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle*, size\_t \* *count* )

Parameters

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

Return values

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

### 62.6.38 void SPDIF\_TransferAbortSend ( SPDIF\_Type \* *base*, spdif\_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>   | SPDIF base pointer.                                                      |
| <i>handle</i> | Pointer to the spdif_handle_t structure which stores the transfer state. |

### 62.6.39 void SPDIF\_TransferAbortReceive ( SPDIF\_Type \* *base*, spdif\_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>   | SPDIF base pointer                                                       |
| <i>handle</i> | Pointer to the spdif_handle_t structure which stores the transfer state. |

### 62.6.40 void SPDIF\_TransferTxHandleIRQ ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle* )

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | SPDIF base pointer.                      |
| <i>handle</i> | Pointer to the spdif_handle_t structure. |

### 62.6.41 void SPDIF\_TransferRxHandleIRQ ( SPDIF\_Type \* *base*, spdif\_handle\_t \* *handle* )

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | SPDIF base pointer.                      |
| <i>handle</i> | Pointer to the spdif_handle_t structure. |

## 62.7 SPDIF eDMA Driver

### 62.7.1 Overview

#### Data Structures

- struct [spdif\\_edma\\_transfer\\_t](#)  
*SPDIF transfer structure. [More...](#)*
- struct [spdif\\_edma\\_handle\\_t](#)  
*SPDIF DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* [spdif\\_edma\\_callback\\_t](#))(SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*SPDIF eDMA transfer callback function for finish and error.*

#### Driver version

- #define [FSL\\_SPDIF\\_EDMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 5))  
*Version 2.0.5.*

#### eDMA Transactional

- void [SPDIF\\_TransferTxCreateHandleEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*dmaLeftHandle, [edma\\_handle\\_t](#) \*dmaRightHandle)  
*Initializes the SPDIF eDMA handle.*
- void [SPDIF\\_TransferRxCreateHandleEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_callback\\_t](#) callback, void \*userData, [edma\\_handle\\_t](#) \*dmaLeftHandle, [edma\\_handle\\_t](#) \*dmaRightHandle)  
*Initializes the SPDIF Rx eDMA handle.*
- [status\\_t](#) [SPDIF\\_TransferSendEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPDIF transfer using DMA.*
- [status\\_t](#) [SPDIF\\_TransferReceiveEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [spdif\\_edma\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPDIF receive using eDMA.*
- void [SPDIF\\_TransferAbortSendEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle)  
*Aborts a SPDIF transfer using eDMA.*
- void [SPDIF\\_TransferAbortReceiveEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle)  
*Aborts a SPDIF receive using eDMA.*
- [status\\_t](#) [SPDIF\\_TransferGetSendCountEDMA](#) (SPDIF\_Type \*base, spdif\_edma\_handle\_t \*handle, [size\\_t](#) \*count)  
*Gets byte count sent by SPDIF.*



- `status_t SPDIF_TransferGetReceiveCountEDMA (SPDIF_Type *base, spdif_edma_handle_t *handle, size_t *count)`  
*Gets byte count received by SPDIF.*

## 62.7.2 Data Structure Documentation

### 62.7.2.1 struct spdif\_edma\_transfer\_t

#### Data Fields

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

#### Field Documentation

- (1) `uint8_t* spdif_edma_transfer_t::leftData`
- (2) `uint8_t* spdif_edma_transfer_t::rightData`
- (3) `size_t spdif_edma_transfer_t::dataSize`

### 62.7.2.2 struct \_spdif\_edma\_handle

#### Data Fields

- `edma_handle_t * dmaLeftHandle`  
*DMA handler for SPDIF left channel.*
- `edma_handle_t * dmaRightHandle`  
*DMA handler for SPDIF right channel.*
- `uint8_t nbytes`  
*eDMA minor byte transfer count initially configured.*
- `uint8_t count`  
*The transfer data count in a DMA request.*
- `uint32_t state`  
*Internal state for SPDIF eDMA transfer.*
- `spdif_edma_callback_t callback`  
*Callback for users while transfer finish or error occurs.*
- `void * userData`  
*User callback parameter.*
- `edma_tcd_t leftTcd [SPDIF_XFER_QUEUE_SIZE+1U]`  
*TCD pool for eDMA transfer.*
- `edma_tcd_t rightTcd [SPDIF_XFER_QUEUE_SIZE+1U]`  
*TCD pool for eDMA transfer.*
- `spdif_edma_transfer_t spdifQueue [SPDIF_XFER_QUEUE_SIZE]`  
*Transfer queue storing queued transfer.*

- `size_t transferSize [SPDIF_XFER_QUEUE_SIZE]`  
*Data bytes need to transfer, left and right are the same, so use one.*
- `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 spdif_edma_handle_t::nbytes`
- (2) `edma_tcd_t spdif_edma_handle_t::leftTcd[SPDIF_XFER_QUEUE_SIZE+1U]`
- (3) `edma_tcd_t spdif_edma_handle_t::rightTcd[SPDIF_XFER_QUEUE_SIZE+1U]`
- (4) `spdif_edma_transfer_t spdif_edma_handle_t::spdifQueue[SPDIF_XFER_QUEUE_SIZE]`
- (5) `volatile uint8_t spdif_edma_handle_t::queueUser`

### 62.7.3 Function Documentation

**62.7.3.1 void SPDIF\_TransferTxCreateHandleEDMA ( SPDIF\_Type \* *base*,  
spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_callback\_t *callback*, void \*  
*userData*, edma\_handle\_t \* *dmaLeftHandle*, edma\_handle\_t \* *dmaRightHandle* )**

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

Parameters

|                        |                                                                                        |
|------------------------|----------------------------------------------------------------------------------------|
| <i>base</i>            | SPDIF base pointer.                                                                    |
| <i>handle</i>          | SPDIF eDMA handle pointer.                                                             |
| <i>base</i>            | SPDIF peripheral base address.                                                         |
| <i>callback</i>        | Pointer to user callback function.                                                     |
| <i>userData</i>        | User parameter passed to the callback function.                                        |
| <i>dmaLeftHandle</i>   | eDMA handle pointer for left channel, this handle shall be static allocated by users.  |
| <i>dmaRight-Handle</i> | eDMA handle pointer for right channel, this handle shall be static allocated by users. |

**62.7.3.2 void SPDIF\_TransferRxCreateHandleEDMA ( SPDIF\_Type \* *base*,  
spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_callback\_t *callback*, void \*  
*userData*, edma\_handle\_t \* *dmaLeftHandle*, edma\_handle\_t \* *dmaRightHandle* )**

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

## Parameters

|                        |                                                                                        |
|------------------------|----------------------------------------------------------------------------------------|
| <i>base</i>            | SPDIF base pointer.                                                                    |
| <i>handle</i>          | SPDIF eDMA handle pointer.                                                             |
| <i>base</i>            | SPDIF peripheral base address.                                                         |
| <i>callback</i>        | Pointer to user callback function.                                                     |
| <i>userData</i>        | User parameter passed to the callback function.                                        |
| <i>dmaLeftHandle</i>   | eDMA handle pointer for left channel, this handle shall be static allocated by users.  |
| <i>dmaRight-Handle</i> | eDMA handle pointer for right channel, this handle shall be static allocated by users. |

### 62.7.3.3 **status\_t SPDIF\_TransferSendEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_transfer\_t \* *xfer* )**

## Note

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

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | SPDIF base pointer.                    |
| <i>handle</i> | SPDIF eDMA handle pointer.             |
| <i>xfer</i>   | Pointer to the DMA transfer structure. |

## Return values

|                                |                                       |
|--------------------------------|---------------------------------------|
| <i>kStatus_Success</i>         | Start a SPDIF eDMA send successfully. |
| <i>kStatus_InvalidArgument</i> | The input argument is invalid.        |
| <i>kStatus_TxBusy</i>          | SPDIF is busy sending data.           |

### 62.7.3.4 **status\_t SPDIF\_TransferReceiveEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, spdif\_edma\_transfer\_t \* *xfer* )**

## Note

This interface returns immediately after the transfer initiates. Call the SPDIF\_GetReceive-RemainingBytes to poll the transfer status and check whether the SPDIF transfer is finished.

## Parameters

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

## Return values

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

### 62.7.3.5 void SPDIF\_TransferAbortSendEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle* )

## Parameters

|               |                            |
|---------------|----------------------------|
| <i>base</i>   | SPDIF base pointer.        |
| <i>handle</i> | SPDIF eDMA handle pointer. |

### 62.7.3.6 void SPDIF\_TransferAbortReceiveEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle* )

## Parameters

|               |                            |
|---------------|----------------------------|
| <i>base</i>   | SPDIF base pointer         |
| <i>handle</i> | SPDIF eDMA handle pointer. |

### 62.7.3.7 status\_t SPDIF\_TransferGetSendCountEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

---

|               |                            |
|---------------|----------------------------|
| <i>base</i>   | SPDIF base pointer.        |
| <i>handle</i> | SPDIF eDMA handle pointer. |
| <i>count</i>  | Bytes count sent by SPDIF. |

Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                   |
| <i>kStatus_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

### 62.7.3.8 **status\_t SPDIF\_TransferGetReceiveCountEDMA ( SPDIF\_Type \* *base*, spdif\_edma\_handle\_t \* *handle*, size\_t \* *count* )**

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | SPDIF base pointer             |
| <i>handle</i> | SPDIF eDMA handle pointer.     |
| <i>count</i>  | Bytes count received by SPDIF. |

Return values

|                                     |                                                   |
|-------------------------------------|---------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                   |
| <i>kStatus_NoTransferInProgress</i> | There is no non-blocking transaction in progress. |

## Chapter 63

# SSARC: State Save and Restore Controller

### 63.1 Overview

The MCUXpresso SDK provides a peripheral driver for the State Save and Restore Controller(SSARC) module of MCUXpresso SDK devices.

The SSARC driver supports the setting of descriptors and groups. For the setting of descriptors, the function [SSARC\\_SetDescriptorConfig\(\)](#) can be used to config the type of descriptors' operation. For the setting of groups, the function [SSARC\\_GroupInit\(\)](#) can be used to config the groups and enable the groups. The function [SSARC\\_GroupDeinit\(\)](#) can be used to disable the groups.

### Data Structures

- struct [ssarc\\_descriptor\\_config\\_t](#)  
*The configuration of descriptor. [More...](#)*
- struct [ssarc\\_group\\_config\\_t](#)  
*The configuration of the group. [More...](#)*

### Enumerations

- enum [\\_ssarc\\_interrupt\\_status\\_flags](#) {  
    [kSSARC\\_AddressErrorFlag](#) = SSARC\_LP\_INT\_STATUS\_ADDR\_ERR\_MASK,  
    [kSSARC\\_AHBErrorFlag](#) = SSARC\_LP\_INT\_STATUS\_AHB\_ERR\_MASK,  
    [kSSARC\\_SoftwareRequestDoneFlag](#) = SSARC\_LP\_INT\_STATUS\_SW\_REQ\_DONE\_MASK,  
    [kSSARC\\_TimeoutFlag](#) = SSARC\_LP\_INT\_STATUS\_TIMEOUT\_MASK,  
    [kSSARC\\_GroupConflictFlag](#) = SSARC\_LP\_INT\_STATUS\_GROUP\_CONFLICT\_MASK }  
*The enumeration of ssarc status flags.*
- enum [ssarc\\_descriptor\\_register\\_size\\_t](#) {  
    [kSSARC\\_DescriptorRegister8bitWidth](#) = 0x0U,  
    [kSSARC\\_DescriptorRegister16bitWidth](#) = 0x1U,  
    [kSSARC\\_DescriptorRegister32bitWidth](#) = 0x2U }  
*The size of the register to be saved/restored.*
- enum [ssarc\\_descriptor\\_operation\\_t](#) {  
    [kSSARC\\_SaveDisableRestoreDisable](#) = 0x0U,  
    [kSSARC\\_SaveEnableRestoreDisable](#) = SSARC\_HP\_SRAM2\_SV\_EN\_MASK,  
    [kSSARC\\_SaveDisableRestoreEnable](#) = SSARC\_HP\_SRAM2\_RT\_EN\_MASK,  
    [kSSARC\\_SaveEnableRestoreEnable](#) = (SSARC\_HP\_SRAM2\_RT\_EN\_MASK | SSARC\_HP\_SRAM2\_SV\_EN\_MASK) }  
*The operation of the descriptor.*
- enum [ssarc\\_descriptor\\_type\\_t](#) {

```

kSSARC_ReadValueWriteBack = 0x00U,
kSSARC_WriteFixedValue = 0x01U,
kSSARC_RMWOOr = 0x02U,
kSSARC_RMWAnd = 0x03U,
kSSARC_DelayCycles = 0x04U,
kSSARC_Polling0 = 0x05U,
kSSARC_Polling1 = 0x06U }

```

*The type of operation.*

- enum `ssarc_save_restore_order_t` {  
`kSSARC_ProcessFromStartToEnd` = 0U,  
`kSSARC_ProcessFromEndToStart` = 1U }

*The order of the restore/save operation.*

- enum `ssarc_software_trigger_mode_t` {  
`kSSARC_TriggerSaveRequest` = SSARC\_LP\_DESC\_CTRL1\_SW\_TRIG\_SV\_MASK,  
`kSSARC_TriggerRestoreRequest` = SSARC\_LP\_DESC\_CTRL1\_SW\_TRIG\_RT\_MASK }

*Software trigger mode.*

## Driver version

- #define `FSL_SSARC_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 0))  
SSARC driver version 2.1.0.

## Descriptor related APIs.

- static uint32\_t `SSARC_GetDescriptorRegisterAddress` (SSARC\_HP\_Type \*base, uint32\_t index)  
*Gets the address of the register to be saved/restored.*
- static uint32\_t `SSARC_GetDescriptorRegisterData` (SSARC\_HP\_Type \*base, uint32\_t index)  
*Gets the value of the register to be saved/restored.*
- void `SSARC_SetDescriptorConfig` (SSARC\_HP\_Type \*base, uint32\_t index, const `ssarc_descriptor_config_t` \*config)  
*Sets the configuration of the descriptor.*

## Group Related APIs

- void `SSARC_GroupInit` (SSARC\_LP\_Type \*base, uint8\_t groupID, const `ssarc_group_config_t` \*config)  
*Initiates the selected group.*
- static void `SSARC_GroupDeinit` (SSARC\_LP\_Type \*base, uint8\_t groupID)  
*De-initiates the selected group.*
- static void `SSARC_LockGroupDomain` (SSARC\_LP\_Type \*base, uint8\_t groupID)  
*Locks the configuration of the domain.*
- static void `SSARC_LockGroupWrite` (SSARC\_LP\_Type \*base, uint8\_t groupID)  
*Locks the write access to the control registers and descriptors for the selected group.*
- static void `SSARC_LockGroupRead` (SSARC\_LP\_Type \*base, uint8\_t groupID)  
*Locks the read access to the control registers and descriptors for the selected group.*
- void `SSARC_TriggerSoftwareRequest` (SSARC\_LP\_Type \*base, uint8\_t groupID, `ssarc_software_trigger_mode_t` mode)  
*Triggers software request.*



## Global Setting Related APIs

- static void [SSARC\\_ResetWholeBlock](#) (SSARC\_LP\_Type \*base)  
*Resets the whole SSARC block by software.*
- static void [SSARC\\_EnableHardwareRequest](#) (SSARC\_LP\_Type \*base, bool enable)  
*Enables/Disables save/restore request from the PGM module.*

## Status Related APIs

- static uint32\_t [SSARC\\_GetStatusFlags](#) (SSARC\_LP\_Type \*base)  
*Gets status flags.*
- static void [SSARC\\_ClearStatusFlags](#) (SSARC\_LP\_Type \*base, uint32\_t mask)  
*Clears status flags.*
- static uint32\_t [SSARC\\_GetErrorIndex](#) (SSARC\_LP\_Type \*base)  
*Gets the error index that indicates which descriptor will trigger the AHB\_ERR or ADDR\_ERR interrupt.*

## Time Out Related APIs

- static void [SSARC\\_SetTimeoutValue](#) (SSARC\_LP\_Type \*base, uint32\_t value)  
*Sets timeout value for the entire group to complete.*
- static uint32\_t [SSARC\\_GetTimeoutValue](#) (SSARC\_LP\_Type \*base)  
*Gets timeout value for AHB clock.*

## Pending Group Related APIs

- static uint16\_t [SSARC\\_GetHardwareRequestRestorePendingGroup](#) (SSARC\_LP\_Type \*base)  
*Gets the value that indicates which groups are pending for restore from hardware request.*
- static uint16\_t [SSARC\\_GetHardwareRequestSavePendingGroup](#) (SSARC\_LP\_Type \*base)  
*Gets the value that indicates which groups are pending for save from hardware request.*
- static uint16\_t [SSARC\\_GetSoftwareRequestRestorePendingGroup](#) (SSARC\_LP\_Type \*base)  
*Gets the value that indicates which groups are pending for restore from software request.*
- static uint16\_t [SSARC\\_GetSoftwareRequestSavePendingGroup](#) (SSARC\_LP\_Type \*base)  
*Gets the value that indicates which groups are pending for save from software request.*

## 63.2 Data Structure Documentation

### 63.2.1 struct ssarc\_descriptor\_config\_t

#### Data Fields

- uint32\_t [address](#)  
*The address of the register/memory to be saved/restored.*
- uint32\_t [data](#)  
*The value of the register/memory to be saved/restored, please note that if the type is selected as kSSARC\_ReadValueWriteBack, this data field is useless.*
- [ssarc\\_descriptor\\_register\\_size\\_t](#) [size](#)  
*The size of register to be saved/restored.*
- [ssarc\\_descriptor\\_operation\\_t](#) [operation](#)  
*The operation mode of descriptor.*
- [ssarc\\_descriptor\\_type\\_t](#) [type](#)  
*The type of operation.*

**Field Documentation**

- (1) `uint32_t ssarc_descriptor_config_t::address`
- (2) `uint32_t ssarc_descriptor_config_t::data`
- (3) `ssarc_descriptor_register_size_t ssarc_descriptor_config_t::size`
- (4) `ssarc_descriptor_operation_t ssarc_descriptor_config_t::operation`
- (5) `ssarc_descriptor_type_t ssarc_descriptor_config_t::type`

**63.2.2 struct ssarc\_group\_config\_t****Data Fields**

- `ssarc_cpu_domain_name_t` [cpuDomain](#)  
*CPU domain, define the ownership of this group.*
- `uint32_t` [startIndex](#)  
*The index of the first descriptor of the group.*
- `uint32_t` [endIndex](#)  
*The index of the last descriptor of the group.*
- `ssarc_save_restore_order_t` [restoreOrder](#)  
*The restore order.*
- `ssarc_save_restore_order_t` [saveOrder](#)  
*The save order.*
- `uint8_t` [restorePriority](#)  
*Restore priority of current group.*
- `uint8_t` [savePriority](#)  
*Save priority of current group.*
- `ssarc_power_domain_name_t` [powerDomain](#)  
*Power domain.*
- `uint32_t` [highestAddress](#)  
*Highest address that can be accessed for the descriptors in the group.*
- `uint32_t` [lowestAddress](#)  
*Lowest address that can be accessed for the descriptors in the group.*

**Field Documentation**

- (1) `ssarc_cpu_domain_name_t ssarc_group_config_t::cpuDomain`
- (2) `uint32_t ssarc_group_config_t::startIndex`
- (3) `uint32_t ssarc_group_config_t::endIndex`
- (4) `ssarc_save_restore_order_t ssarc_group_config_t::restoreOrder`
- (5) `ssarc_save_restore_order_t ssarc_group_config_t::saveOrder`

### (6) `uint8_t ssarc_group_config_t::restorePriority`

0 is the highest priority, 15 is the lowest priority

### (7) `uint8_t ssarc_group_config_t::savePriority`

0 is the highest priority, 15 is the lowest priority.

### (8) `ssarc_power_domain_name_t ssarc_group_config_t::powerDomain`

### (9) `uint32_t ssarc_group_config_t::highestAddress`

### (10) `uint32_t ssarc_group_config_t::lowestAddress`

## 63.3 Macro Definition Documentation

### 63.3.1 `#define FSL_SSARC_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

## 63.4 Enumeration Type Documentation

### 63.4.1 `enum _ssarc_interrupt_status_flags`

Enumerator

- kSSARC\_AddressErrorFlag* If the descriptor is not in the range, assert address error.
- kSSARC\_AHBErrorFlag* If any AHB master access receives none-OKAY, assert AHB error.
- kSSARC\_SoftwareRequestDoneFlag* If a software triggered save or restore process is completed, assert software request done .
- kSSARC\_TimeoutFlag* If processing of a group has exceeded the timeout value, assert timeout.
- kSSARC\_GroupConflictFlag* Group conflict.

### 63.4.2 `enum ssarc_descriptor_register_size_t`

Enumerator

- kSSARC\_DescriptorRegister8bitWidth* The register to be saved/restored is 8 bit width.
- kSSARC\_DescriptorRegister16bitWidth* The register to be saved/restored is 16 bit width.
- kSSARC\_DescriptorRegister32bitWidth* The register to be saved/restored is 32 bit width.

### 63.4.3 `enum ssarc_descriptor_operation_t`

Enumerator

- kSSARC\_SaveDisableRestoreDisable* Disable Save operation, disable restore operation.
- kSSARC\_SaveEnableRestoreDisable* Enable Save operation, disable restore operation.

***kSSARC\_SaveDisableRestoreEnable*** Disable Save operation, enable restore operation.

***kSSARC\_SaveEnableRestoreEnable*** Enable Save operation, enable restore operation.

#### 63.4.4 enum ssarc\_descriptor\_type\_t

Enumerator

***kSSARC\_ReadValueWriteBack*** Read the register value on save operation and write it back on restore operation.

***kSSARC\_WriteFixedValue*** Always write a fixed value from DATA[31:0].

***kSSARC\_RMWO*** Read register, OR with the DATA[31:0], and write it back.

***kSSARC\_RMWAnd*** Read register, AND with the DATA[31:0], and write it back.

***kSSARC\_DelayCycles*** Delay for number of cycles based on the DATA[31:0].

***kSSARC\_Polling0*** Read the register until read\_data[31:0] & DATA[31:0] == 0.

***kSSARC\_Polling1*** Read the register until read\_data[31:0] & DATA[31:0] != 0.

#### 63.4.5 enum ssarc\_save\_restore\_order\_t

Enumerator

***kSSARC\_ProcessFromStartToEnd*** Descriptors within the group are processed from start to end.

***kSSARC\_ProcessFromEndToStart*** Descriptors within the group are processed from end to start.

#### 63.4.6 enum ssarc\_software\_trigger\_mode\_t

Enumerator

***kSSARC\_TriggerSaveRequest*** Don't trigger restore operation, trigger the save operation by software.

***kSSARC\_TriggerRestoreRequest*** Trigger the restore operation, don't trigger the save operation.

### 63.5 Function Documentation

**63.5.1 static uint32\_t SSARC\_GetDescriptorRegisterAddress ( SSARC\_HP\_Type \*  
base, uint32\_t index ) [inline], [static]**

## Parameters

|              |                                                |
|--------------|------------------------------------------------|
| <i>base</i>  | SSARC_HP peripheral base address.              |
| <i>index</i> | The index of descriptor. Range from 0 to 1023. |

## Returns

The address of the register.

**63.5.2 static uint32\_t SSARC\_GetDescriptorRegisterData ( SSARC\_HP\_Type \* *base*, uint32\_t *index* ) [inline], [static]**

## Parameters

|              |                                                |
|--------------|------------------------------------------------|
| <i>base</i>  | SSARC_HP peripheral base address.              |
| <i>index</i> | The index of descriptor. Range from 0 to 1023. |

## Returns

The value of the register.

**63.5.3 void SSARC\_SetDescriptorConfig ( SSARC\_HP\_Type \* *base*, uint32\_t *index*, const ssarc\_descriptor\_config\_t \* *config* )**

## Parameters

|               |                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SSARC_HP peripheral base address.                                                                                                           |
| <i>index</i>  | The index of descriptor. Range from 0 to 1023.                                                                                              |
| <i>config</i> | Pointer to the structure <a href="#">ssarc_descriptor_config_t</a> . Please refer to <a href="#">ssarc_descriptor_config_t</a> for details. |

**63.5.4 void SSARC\_GroupInit ( SSARC\_LP\_Type \* *base*, uint8\_t *groupID*, const ssarc\_group\_config\_t \* *config* )**

## Note

For the groups with the same save priority or restore priority, the save/restore operation runs in the group order.

## Parameters

|                |                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | SSARC_LP peripheral base address.                                                                                                 |
| <i>groupID</i> | The index of the group. Range from 0 to 15.                                                                                       |
| <i>config</i>  | Pointer to the structure <a href="#">ssarc_group_config_t</a> . Please refer to <a href="#">ssarc_group_config_t</a> for details. |

### 63.5.5 static void SSARC\_GroupDeinit ( SSARC\_LP\_Type \* *base*, uint8\_t *groupID* ) [inline], [static]

## Parameters

|                |                                             |
|----------------|---------------------------------------------|
| <i>base</i>    | SSARC_LP peripheral base address.           |
| <i>groupID</i> | The index of the group. Range from 0 to 15. |

### 63.5.6 static void SSARC\_LockGroupDomain ( SSARC\_LP\_Type \* *base*, uint8\_t *groupID* ) [inline], [static]

This function locks the configuration of the domain. Once locked, only the access from the same domain is allowed, access from other domains will be blocked. Once locked, it can only be unlocked by a hardware reset.

## Parameters

|                |                                             |
|----------------|---------------------------------------------|
| <i>base</i>    | SSARC_LP peripheral base address.           |
| <i>groupID</i> | The index of the group. Range from 0 to 15. |

### 63.5.7 static void SSARC\_LockGroupWrite ( SSARC\_LP\_Type \* *base*, uint8\_t *groupID* ) [inline], [static]

This function Locks the write access to the control registers and descriptors for the selected group. All writes are blocked. Once locked, it can only be unlocked by a hardware reset.

## Parameters

---

|                |                                             |
|----------------|---------------------------------------------|
| <i>base</i>    | SSARC_LP peripheral base address.           |
| <i>groupID</i> | The index of the group. Range from 0 to 15. |

### 63.5.8 static void SSARC\_LockGroupRead ( SSARC\_LP\_Type \* *base*, uint8\_t *groupID* ) [inline], [static]

This function Locks the read access to the control registers and descriptors for the selected group. All reads are blocked. Once locked, it can only be unlocked by a hardware reset.

Parameters

|                |                                             |
|----------------|---------------------------------------------|
| <i>base</i>    | SSARC_LP peripheral base address.           |
| <i>groupID</i> | The index of the group. Range from 0 to 15. |

### 63.5.9 void SSARC\_TriggerSoftwareRequest ( SSARC\_LP\_Type \* *base*, uint8\_t *groupID*, ssarc\_software\_trigger\_mode\_t *mode* )

Note

Each group allows software to trigger the save/restore operation without getting the request from basic power controller.

Parameters

|                |                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>    | SSARC_LP peripheral base address.                                                                 |
| <i>groupID</i> | The index of the group. Range from 0 to 15.                                                       |
| <i>mode</i>    | Software trigger mode. Please refer to <a href="#">ssarc_software_trigger_mode_t</a> for details. |

### 63.5.10 static void SSARC\_ResetWholeBlock ( SSARC\_LP\_Type \* *base* ) [inline], [static]

Note

Only reset the SSARC registers, not include the DESC in SRAM.

## Parameters

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

**63.5.11 static void SSARC\_EnableHardwareRequest ( SSARC\_LP\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

|               |                                                                                                                                                                                                                    |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | SSARC_LP peripheral base address.                                                                                                                                                                                  |
| <i>enable</i> | Used to enable/disable save/restore hardware request. <ul style="list-style-type: none"> <li>• <b>true</b> Enable GPC save/restore requests.</li> <li>• <b>false</b> Disable GPC save/restore requests.</li> </ul> |

**63.5.12 static uint32\_t SSARC\_GetStatusFlags ( SSARC\_LP\_Type \* *base* ) [inline], [static]**

## Parameters

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

## Returns

The value of status flags. See [\\_ssarc\\_interrupt\\_status\\_flags](#) for details.

**63.5.13 static void SSARC\_ClearStatusFlags ( SSARC\_LP\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Note

Only [kSSARC\\_AddressErrorFlag](#), [kSSARC\\_AHBErrorFlag](#), [kSSARC\\_TimeoutFlag](#) and [kSSARC\\_GroupConflictFlag](#) can be cleared.



## Parameters

|             |                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------|
| <i>base</i> | SSARC_LP peripheral base address.                                                                      |
| <i>mask</i> | The mask value for flags to be cleared. See <a href="#">_ssarc_interrupt_status_flags</a> for details. |

**63.5.14** `static uint32_t SSARC_GetErrorIndex ( SSARC_LP_Type * base )  
[inline], [static]`

## Parameters

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

## Returns

The error index.

**63.5.15** `static void SSARC_SetTimeoutValue ( SSARC_LP_Type * base, uint32_t  
value ) [inline], [static]`

This function sets timeout value for the entire group to complete. Setting timeout value to 0 will disable this feature.

## Parameters

|              |                                                      |
|--------------|------------------------------------------------------|
| <i>base</i>  | SSARC_LP peripheral base address.                    |
| <i>value</i> | The timeout value, 0 means disable time out feature. |

**63.5.16** `static uint32_t SSARC_GetTimeoutValue ( SSARC_LP_Type * base )  
[inline], [static]`

## Parameters

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

## Returns

The timeout value.

**63.5.17** `static uint16_t SSARC_GetHardwareRequestRestorePendingGroup (`  
`SSARC_LP_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The value of the pending groups.

**63.5.18** `static uint16_t SSARC_GetHardwareRequestSavePendingGroup (SSARC_LP_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The value of the pending groups.

**63.5.19** `static uint16_t SSARC_GetSoftwareRequestRestorePendingGroup (SSARC_LP_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The value of the pending groups.

**63.5.20** `static uint16_t SSARC_GetSoftwareRequestSavePendingGroup (SSARC_LP_Type * base ) [inline], [static]`

## Parameters

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

## Returns

The value of the pending groups.

## Chapter 64

# TEMPSENSOR: Temperature Sensor Module

### 64.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Temperature Sensor Module (TEMPSENSOR) module of MCUXpresso SDK devices.

### 64.2 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 [tmplsns\\_config\\_t](#)  
*TMPSNS temperature structure. [More...](#)*

### Enumerations

- enum {  
    [kTEMPSENSOR\\_HighTempInterruptStatusEnable](#),  
    [kTEMPSENSOR\\_LowTempInterruptStatusEnable](#),  
    [kTEMPSENSOR\\_PanicTempInterruptStatusEnable](#),  
    [kTEMPSENSOR\\_FinishInterruptStatusEnable](#) }  
    *TMPSNS interrupt status type, tmplsns\_interrupt\_status\_enable\_t.*
- enum [tmplsns\\_measure\\_mode\\_t](#) {  
    [kTEMPSENSOR\\_SingleMode](#) = 0U,  
    [kTEMPSENSOR\\_ContinuousMode](#) = 1U }  
    *TMPSNS measure mode, tempsensor\_measure\_mode.*

- enum `tmplsns_alarm_mode_t` {  
`kTEMPMON_HighAlarmMode` = 0U,  
`kTEMPMON_PanicAlarmMode` = 1U,  
`kTEMPMON_LowAlarmMode` = 2U }  
*TMPSNS alarm mode.*

## Functions

- void `TMPSNS_Init` (TMPSNS\_Type \*base, const `tmplsns_config_t` \*config)  
*Initializes the TMPSNS module.*
- void `TMPSNS_Deinit` (TMPSNS\_Type \*base)  
*Deinitializes the TMPSNS module.*
- void `TMPSNS_GetDefaultConfig` (`tmplsns_config_t` \*config)  
*Gets the default configuration structure.*
- void `TMPSNS_StartMeasure` (TMPSNS\_Type \*base)  
*start the temperature measurement process.*
- void `TMPSNS_StopMeasure` (TMPSNS\_Type \*base)  
*stop the measurement process.*
- float `TMPSNS_GetCurrentTemperature` (TMPSNS\_Type \*base)  
*Get current temperature with the fused temperature calibration data.*
- void `TMPSNS_SetTempAlarm` (TMPSNS\_Type \*base, int32\_t tempVal, `tmplsns_alarm_mode_t` alarmMode)  
*Set the temperature count (raw sensor output) that will generate an alarm interrupt.*
- void `TMPSNS_EnableInterrupt` (TMPSNS\_Type \*base, uint32\_t mask)  
*Enable interrupt status.*
- void `TMPSNS_DisableInterrupt` (TMPSNS\_Type \*base, uint32\_t mask)  
*Disable interrupt status.*

## Driver version

- #define `FSL_TMPSNS_DRIVER_VERSION` (`MAKE_VERSION`(2, 0, 3))  
*Version 2.0.3.*

## 64.3 Data Structure Documentation

### 64.3.1 struct `tmplsns_config_t`

#### Data Fields

- `tmplsns_measure_mode_t` `measureMode`  
*The temperature measure mode.*
- uint16\_t `frequency`  
*The temperature measure frequency.*
- int32\_t `highAlarmTemp`  
*The high alarm temperature.*
- int32\_t `panicAlarmTemp`  
*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`

## 64.4 Enumeration Type Documentation

## 64.4.1 anonymous enum

Enumerator

*kTEMPSENSOR\_HighTempInterruptStatusEnable* High temperature interrupt status enable.  
*kTEMPSENSOR\_LowTempInterruptStatusEnable* High temperature interrupt status enable.  
*kTEMPSENSOR\_PanicTempInterruptStatusEnable* High temperature interrupt status enable.  
*kTEMPSENSOR\_FinishInterruptStatusEnable* High temperature interrupt status enable.

## 64.4.2 enum tmpsns\_measure\_mode\_t

Enumerator

*kTEMPSENSOR\_SingleMode* Single measurement mode.  
*kTEMPSENSOR\_ContinuousMode* Continuous measurement mode.

## 64.4.3 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.

## 64.5 Function Documentation

64.5.1 void TMPSENS\_Init ( TMPSENS\_Type \* *base*, const tmpsns\_config\_t \* *config* )

## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | TMPSNS base pointer                 |
| <i>config</i> | Pointer to configuration structure. |

**64.5.2 void TMPSNS\_Deinit ( TMPSNS\_Type \* *base* )**

## Parameters

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

**64.5.3 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. |
|---------------|---------------------------------------|

**64.5.4 void TMPSNS\_StartMeasure ( TMPSNS\_Type \* *base* )**

## Parameters

|             |                      |
|-------------|----------------------|
| <i>base</i> | TMPSNS base pointer. |
|-------------|----------------------|

**64.5.5 void TMPSNS\_StopMeasure ( TMPSNS\_Type \* *base* )**

## Parameters

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

**64.5.6 float TMPSNS\_GetCurrentTemperature ( TMPSNS\_Type \* *base* )**



## Parameters

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

## Returns

current temperature with degrees Celsius.

#### 64.5.7 void TMPSNS\_SetTempAlarm ( TMPSNS\_Type \* *base*, int32\_t *tempVal*, tmpsns\_alarm\_mode\_t *alarmMode* )

## Parameters

|                  |                                            |
|------------------|--------------------------------------------|
| <i>base</i>      | TMPSNS base pointer                        |
| <i>tempVal</i>   | The alarm temperature with degrees Celsius |
| <i>alarmMode</i> | The alarm mode.                            |

#### 64.5.8 void TMPSNS\_EnableInterrupt ( TMPSNS\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                 |
|-------------|-----------------------------------------------------------------|
| <i>base</i> | TMPSNS base pointer                                             |
| <i>mask</i> | The interrupts to enable from tmpsns_interrupt_status_enable_t. |

#### 64.5.9 void TMPSNS\_DisableInterrupt ( TMPSNS\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                                                                  |
|-------------|------------------------------------------------------------------|
| <i>base</i> | TMPSNS base pointer                                              |
| <i>mask</i> | The interrupts to disable from tmpsns_interrupt_status_enable_t. |

## Chapter 65

# USDHC: Ultra Secured Digital Host Controller Driver

### 65.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Ultra Secured Digital Host Controller (USDHC) module of MCUXpresso SDK/i.MX devices.

### 65.2 Typical use case

#### 65.2.1 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 suppport 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\\_DMADDataAddrNotAlign](#) = 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_CIHCB_MASK,
kUSDHC_DataInhibitFlag = USDHC_PRES_STATE_CDIHB_MASK,
kUSDHC_DataLineActiveFlag = USDHC_PRES_STATE_DLA_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_DLSSL_SHIFT,
kUSDHC_Data1LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 1U),
kUSDHC_Data2LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 2U),
kUSDHC_Data3LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 3U),
kUSDHC_Data4LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 4U),
kUSDHC_Data5LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 5U),
kUSDHC_Data6LineLevelFlag = 1U << (USDHC_PRES_STATE_DLSSL_SHIFT + 6U),
kUSDHC_Data7LineLevelFlag = (int)(1U << (USDHC_PRES_STATE_DLSSL_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_AdmaLenghMismatchFlag = USDHC_ADMA_ERR_STATUS_ADMALME_MASK,
 kUSDHC_AdmaDescriptorErrorFlag = USDHC_ADMA_ERR_STATUS_ADMADCE_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_FEVTC12TOE_MASK,
 kUSDHC_ForceEventCommandCrcError = USDHC_FORCE_EVENT_FEVTC12CE_MASK,
 kUSDHC_ForceEventCommandEndBitError = USDHC_FORCE_EVENT_FEVTC12EBE_MASK,
 kUSDHC_ForceEventCommandIndexError = USDHC_FORCE_EVENT_FEVTC12IE_MASK,
 kUSDHC_ForceEventDataTimeout = USDHC_FORCE_EVENT_FEVTD12TOE_MASK,
 kUSDHC_ForceEventDataCrcError = USDHC_FORCE_EVENT_FEVTD12CE_MASK,
 kUSDHC_ForceEventDataEndBitError = USDHC_FORCE_EVENT_FEVTD12EBE_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\_MASK }
- 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_ResponseTypeNone = 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 | kUSDH-
C_Adma1DescriptorValidFlag),
kUSDHC_Adma1DescriptorTypeLink,
kUSDHC_Adma1DescriptorTypeSetLength = (kUSDHC_Adma1DescriptorActivity1Flag | kUSD-
HC_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 | kUSD-
HC_Adma2DescriptorValidFlag),
kUSDHC_Adma2DescriptorTypeTransfer = (kUSDHC_Adma2DescriptorActivity2Flag | kUSDH-
C_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, 1U))  
     Driver version 2.8.1.

## 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\\_EnableDDRMMode](#) (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\\_SetStrobeDllOverride](#) (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.*

## 65.3 Data Structure Documentation

### 65.3.1 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](#)

### 65.3.2 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`

**65.3.3 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`

### 65.3.4 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.

### 65.3.5 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 reponse.*
- 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`

**65.3.6 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`

**65.3.7 struct usdhc\_scatter\_gather\_data\_list\_t**

Allow application register uncontinuous data buffer for data transfer.

**65.3.8 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](#)

## 65.3.9 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](#)

## 65.3.10 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`

## 65.3.11 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`

### 65.3.12 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)`

### 65.3.13 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****65.3.14 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****65.4 Macro Definition Documentation****65.4.1 #define FSL\_USDHC\_DRIVER\_VERSION (MAKE\_VERSION(2U, 8U, 1U))**

**65.4.2 #define USDHC\_ADMA1\_ADDRESS\_ALIGN (4096U)**

**65.4.3 #define USDHC\_ADMA1\_LENGTH\_ALIGN (4096U)**

**65.4.4 #define USDHC\_ADMA2\_ADDRESS\_ALIGN (4U)**

**65.4.5 #define USDHC\_ADMA2\_LENGTH\_ALIGN (4U)**

**65.4.6 #define USDHC\_ADMA1\_DESCRIPTOR\_ADDRESS\_SHIFT (12U)**

| Address/page field           | Reserved | 6p(*8-*3)*6/8lightgrayAttribute |      |    |     |   |
|------------------------------|----------|---------------------------------|------|----|-----|---|
| 31 12                        | 11 6     | 05                              | 04   | 03 | 02  | 0 |
| address<br>or data<br>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**

**65.4.7 #define USDHC\_ADMA1\_DESCRIPTOR\_ADDRESS\_MASK (0xFFFFFU)**

**65.4.8 #define USDHC\_ADMA1\_DESCRIPTOR\_LENGTH\_SHIFT (12U)**

**65.4.9 #define USDHC\_ADMA1\_DESCRIPTOR\_LENGTH\_MASK (0xFFFFFU)**

#### 65.4.10 #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.

#### 65.4.11 #define USDHC\_ADMA2\_DESCRIPTOR\_LENGTH\_SHIFT (16U)

| Address field  | Length        | Reserved   | Attribute |      |    |     |
|----------------|---------------|------------|-----------|------|----|-----|
| 63 32          | 31 16         | 15 06      | 05        | 04   | 03 | 02  |
| 32-bit address | 16-bit length | 0000000000 | Act2      | Act1 | 0  | Int |

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

#### 65.4.12 #define USDHC\_ADMA2\_DESCRIPTOR\_LENGTH\_MASK (0xFFFFU)

#### 65.4.13 #define USDHC\_ADMA2\_DESCRIPTOR\_MAX\_LENGTH\_PER\_ENTRY (USDHC\_ADMA2\_DESCRIPTOR\_LENGTH\_MASK - 3U)

### 65.5 Typedef Documentation

**65.5.1** `typedef uint32_t usdhc_adma1_descriptor_t`

**65.5.2** `typedef status_t(* usdhc_transfer_function_t)(USDHC_Type *base,  
usdhc_transfer_t *content)`

## 65.6 Enumeration Type Documentation

### 65.6.1 anonymous enum

USDHC status.

Enumerator

*kStatus\_USDHC\_BusyTransferring* Transfer is on-going.  
*kStatus\_USDHC\_PrepareAdmaDescriptorFailed* Set DMA descriptor failed.  
*kStatus\_USDHC\_SendCommandFailed* Send command failed.  
*kStatus\_USDHC\_TransferDataFailed* Transfer data failed.  
*kStatus\_USDHC\_DMADDataAddrNotAlign* 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.

### 65.6.2 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.

### 65.6.3 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.

### 65.6.4 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.

### 65.6.5 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.



### 65.6.6 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.

### 65.6.7 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.

### 65.6.8 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.

### 65.6.9 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 sampling clock.

### 65.6.10 anonymous enum

ADMA error status flag mask.

Enumerator

***kUSDHC\_AdmaLenghMismatchFlag*** Length mismatch error.

***kUSDHC\_AdmaDescriptorErrorFlag*** Descriptor error.

### 65.6.11 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.

### 65.6.12 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.

### 65.6.13 enum usdhc\_transfer\_direction\_t

Enumerator

***kUSDHC\_TransferDirectionReceive*** USDHC transfer direction receive.  
***kUSDHC\_TransferDirectionSend*** USDHC transfer direction send.

### 65.6.14 enum usdhc\_data\_bus\_width\_t

Enumerator

***kUSDHC\_DataBusWidth1Bit*** 1-bit mode  
***kUSDHC\_DataBusWidth4Bit*** 4-bit mode  
***kUSDHC\_DataBusWidth8Bit*** 8-bit mode

### 65.6.15 enum usdhc\_endian\_mode\_t

Enumerator

***kUSDHC\_EndianModeBig*** Big endian mode.  
***kUSDHC\_EndianModeHalfWordBig*** Half word big endian mode.  
***kUSDHC\_EndianModeLittle*** Little endian mode.

### 65.6.16 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.

### 65.6.17 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.

### 65.6.18 enum usdhc\_boot\_mode\_t

Enumerator

*kUSDHC\_BootModeNormal* Normal boot.  
*kUSDHC\_BootModeAlternative* Alternative boot.

### 65.6.19 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.

### 65.6.20 enum usdhc\_card\_response\_type\_t

Defines the command response type from card to host controller.

Enumerator

*kCARD\_ResponseTypeNone* Response type: none.  
*kCARD\_ResponseTypeR1* Response type: R1.  
*kCARD\_ResponseTypeR1b* Response type: R1b.  
*kCARD\_ResponseTypeR2* Response type: R2.  
*kCARD\_ResponseTypeR3* Response type: R3.  
*kCARD\_ResponseTypeR4* Response type: R4.  
*kCARD\_ResponseTypeR5* Response type: R5.

***kCARD\_ResponseTypeR5b*** Response type: R5b.

***kCARD\_ResponseTypeR6*** Response type: R6.

***kCARD\_ResponseTypeR7*** Response type: R7.

### 65.6.21 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.

### 65.6.22 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.

### 65.6.23 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.

### 65.6.24 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.

### 65.6.25 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.

## 65.7 Function Documentation

### 65.7.1 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. |
|------------------------|-----------------------|

### 65.7.2 void USDHC\_Deinit ( USDHC\_Type \* *base* )

Parameters

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

### 65.7.3 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( <a href="#">_usdhc_reset</a> ). |
| <i>timeout</i> | Timeout for reset.                                   |

Return values

|              |                     |
|--------------|---------------------|
| <i>true</i>  | Reset successfully. |
| <i>false</i> | Reset failed.       |

### 65.7.4 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 <a href="#">_usdhc_adma_flag</a> . |

## Return values

|                                    |                                                             |
|------------------------------------|-------------------------------------------------------------|
| <a href="#">kStatus_OutOfRange</a> | ADMA descriptor table length isn't enough to describe data. |
| <a href="#">kStatus_Success</a>    | Operate successfully.                                       |

**65.7.5 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

|                                    |                                                             |
|------------------------------------|-------------------------------------------------------------|
| <a href="#">kStatus_OutOfRange</a> | ADMA descriptor table length isn't enough to describe data. |
| <a href="#">kStatus_Success</a>    | Operate successfully.                                       |

**65.7.6 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>admaTable-<br/>Words</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 <a href="#">_usdhc_adma_flag</a> . |

## Return values

|                                           |                                                             |
|-------------------------------------------|-------------------------------------------------------------|
| <a href="#"><i>kStatus_OutOfRange</i></a> | ADMA descriptor table length isn't enough to describe data. |
| <a href="#"><i>kStatus_Success</i></a>    | Operate successfully.                                       |

**65.7.7** `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>admaTable-<br/>Words</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 <a href="#">_usdhc_adma_flag</a> . |

## Return values

|                                           |                                                             |
|-------------------------------------------|-------------------------------------------------------------|
| <a href="#"><i>kStatus_OutOfRange</i></a> | ADMA descriptor table length isn't enough to describe data. |
| <a href="#"><i>kStatus_Success</i></a>    | Operate successfully.                                       |

**65.7.8** `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         |

**65.7.9 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( <a href="#">_usdhc_interrupt_status_flag</a> ). |

**65.7.10 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( <a href="#">_usdhc_interrupt_status_flag</a> ). |

**65.7.11 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( <a href="#">_usdhc_interrupt_status_flag</a> ). |

**65.7.12 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( <a href="#">_usdhc_interrupt_status_flag</a> ). |

**65.7.13** `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](#)).

**65.7.14** `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](#)).

**65.7.15** `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( <a href="#">_usdhc_interrupt_status_flag</a> ). |

#### 65.7.16 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](#)).

#### 65.7.17 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](#)).

#### 65.7.18 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](#)).

**65.7.19** 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. |

### 65.7.20 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            |

### 65.7.21 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.

### 65.7.22 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.       |

**65.7.23 static void USDHC\_AssertHardwareReset ( USDHC\_Type \* *base*, bool *high* ) [inline], [static]**

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USDHC peripheral base address. |
| <i>high</i> | 1 or 0 level                   |

**65.7.24 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.           |

**65.7.25 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.     |



**65.7.26** `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.

**65.7.27 void USDHC\_SendCommand ( USDHC\_Type \* *base*, usdhc\_command\_t \* *command* )**

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | USDHC peripheral base address. |
| <i>command</i> | configuration                  |

**65.7.28 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( <a href="#">_usdhc_wakeup_event</a> ). |
| <i>enable</i> | True to enable, false to disable.                          |

**65.7.29 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            |

**65.7.30 static bool USDHC\_DetectCardInsert ( USDHC\_Type \* *base* ) [inline], [static]**

## Parameters

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

**65.7.31** `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( <a href="#">_usdhc_sdio_control_flag</a> ). |
| <i>enable</i> | True to enable, false to disable.                                         |

**65.7.32** `static void USDHC_SetContinueRequest ( USDHC_Type * base ) [inline], [static]`

## Parameters

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

**65.7.33** `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. |

**65.7.34** `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. |

### 65.7.35 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. |

### 65.7.36 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). |

### 65.7.37 static void USDHC\_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. |

### 65.7.38 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. |
|-------------|--------------------------------|

**65.7.39 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. |
|-------------|--------------------------------|

**65.7.40 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            |

**65.7.41 void USDHC\_EnableAutoTuningForCmdAndData ( USDHC\_Type \* *base* )**

Parameters

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

**65.7.42 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             |

**65.7.43** `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. |
|------------|-------------------------------------------|

**65.7.44** `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 |

**65.7.45** `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 |

**65.7.46 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 |

**65.7.47 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 |
|---------------|---------------------|

**65.7.48** `static uint32_t USDHC_GetExecuteStdTuningStatus ( USDHC_Type * base ) [inline], [static]`

Parameters

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

**65.7.49** `static uint32_t USDHC_CheckStdTuningResult ( USDHC_Type * base ) [inline], [static]`

Parameters

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

**65.7.50** `static uint32_t USDHC_CheckTuningError ( USDHC_Type * base ) [inline], [static]`

Parameters

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

**65.7.51** `void USDHC_EnableDDRMMode ( 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                |



```
65.7.52 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            |

**65.7.53 static void USDHC\_ResetStrobeDLL ( USDHC\_Type \* *base* ) [inline], [static]**

## Parameters

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

**65.7.54 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            |

**65.7.55 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                |

**65.7.56 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. |

**65.7.57** `static uint32_t USDHC_GetStrobeDLLStatus ( USDHC_Type * base )  
[inline], [static]`

## Parameters

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

**65.7.58** `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.               |

**65.7.59** `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.                         |

**65.7.60** `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

|                                                                    |                                 |
|--------------------------------------------------------------------|---------------------------------|
| <a href="#"><i>kStatus_InvalidArgument</i></a>                     | Argument is invalid.            |
| <a href="#"><i>kStatus_USDHC_Busy-Transferring</i></a>             | Busy transferring.              |
| <a href="#"><i>kStatus_USDHC_-PrepareAdmaDescriptor-Failed</i></a> | Prepare ADMA descriptor failed. |
| <a href="#"><i>kStatus_Success</i></a>                             | Operate successfully.           |

**65.7.61** `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_-PrepareAdmaDescriptor-Failed</i> | Prepare ADMA descriptor failed. |
| <i>kStatus_USDHC_Send-CommandFailed</i>            | Send command failed.            |
| <i>kStatus_USDHC_-TransferDataFailed</i>           | Transfer data failed.           |
| <i>kStatus_Success</i>                             | Operate successfully.           |

### 65.7.62 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 66

# WDOG: Watchdog Timer Driver

### 66.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

### 66.2 Typical use case

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

### Data Structures

- struct [wdog\\_work\\_mode\\_t](#)  
*Defines WDOG work mode. [More...](#)*
- struct [wdog\\_config\\_t](#)  
*Describes WDOG configuration structure. [More...](#)*

### Enumerations

- enum [\\_wdog\\_interrupt\\_enable](#) { [kWDOG\\_InterruptEnable](#) = WDOG\_WICR\_WIE\_MASK }
- enum [\\_wdog\\_status\\_flags](#) {  
[kWDOG\\_RunningFlag](#) = WDOG\_WCR\_WDE\_MASK,  
[kWDOG\\_PowerOnResetFlag](#) = WDOG\_WRSR\_POR\_MASK,  
[kWDOG\\_TimeoutResetFlag](#) = WDOG\_WRSR\_TOUT\_MASK,  
[kWDOG\\_SoftwareResetFlag](#) = WDOG\_WRSR\_SFTW\_MASK,  
[kWDOG\\_InterruptFlag](#) = WDOG\_WICR\_WTIS\_MASK }
- WDOG status flags.*

### Driver version

- #define [FSL\\_WDOG\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))  
*Defines WDOG driver version.*

### Refresh sequence

- #define [WDOG\\_REFRESH\\_KEY](#) (0xAAAA5555U)

### WDOG Initialization and De-initialization.

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

- *Initializes the WDOG.*
- void [WDOG\\_Deinit](#) (WDOG\_Type \*base)
- *Shuts down the WDOG.*
- static void [WDOG\\_Enable](#) (WDOG\_Type \*base)
- *Enables the WDOG module.*
- static void [WDOG\\_Disable](#) (WDOG\_Type \*base)
- *Disables the WDOG module.*
- static void [WDOG\\_TriggerSystemSoftwareReset](#) (WDOG\_Type \*base)
- *Trigger the system software reset.*
- static void [WDOG\\_TriggerSoftwareSignal](#) (WDOG\_Type \*base)
- *Trigger an output assertion.*
- static void [WDOG\\_EnableInterrupts](#) (WDOG\_Type \*base, uint16\_t mask)
- *Enables the WDOG interrupt.*
- uint16\_t [WDOG\\_GetStatusFlags](#) (WDOG\_Type \*base)
- *Gets the WDOG all reset status flags.*
- void [WDOG\\_ClearInterruptStatus](#) (WDOG\_Type \*base, uint16\_t mask)
- *Clears the WDOG flag.*
- static void [WDOG\\_SetTimeoutValue](#) (WDOG\_Type \*base, uint16\_t timeoutCount)
- *Sets the WDOG timeout value.*
- static void [WDOG\\_SetInterruptTimeoutValue](#) (WDOG\_Type \*base, uint16\_t timeoutCount)
- *Sets the WDOG interrupt count timeout value.*
- static void [WDOG\\_DisablePowerDownEnable](#) (WDOG\_Type \*base)
- *Disable the WDOG power down enable bit.*
- void [WDOG\\_Refresh](#) (WDOG\_Type \*base)
- *Refreshes the WDOG timer.*

## 66.3 Data Structure Documentation

### 66.3.1 struct wdog\_work\_mode\_t

#### Data Fields

- bool [enableWait](#)  
*continue or suspend WDOG in wait mode*
- bool [enableStop](#)  
*continue or suspend WDOG in stop mode*
- bool [enableDebug](#)  
*continue or suspend WDOG in debug mode*

### 66.3.2 struct wdog\_config\_t

#### Data Fields

- bool [enableWdog](#)  
*Enables or disables WDOG.*
- [wdog\\_work\\_mode\\_t](#) [workMode](#)  
*Configures WDOG work mode in debug stop and wait mode.*
- bool [enableInterrupt](#)

- *Enables or disables WDOG interrupt.*  
uint16\_t **timeoutValue**  
*Timeout value.*
- uint16\_t **interruptTimeValue**  
*Interrupt count timeout value.*
- bool **softwareResetExtension**  
*software reset extension*
- bool **enablePowerDown**  
*power down enable bit*
- bool **enableTimeOutAssert**  
*Enable WDOG\_B timeout assertion.*

## Field Documentation

(1) bool wdog\_config\_t::enableTimeOutAssert

## 66.4 Enumeration Type Documentation

### 66.4.1 enum \_wdog\_interrupt\_enable

This structure contains the settings for all of the WDOG interrupt configurations.

Enumerator

***kWDOG\_InterruptEnable*** WDOG timeout generates an interrupt before reset.

### 66.4.2 enum \_wdog\_status\_flags

This structure contains the WDOG status flags for use in the WDOG functions.

Enumerator

***kWDOG\_RunningFlag*** Running flag, set when WDOG is enabled.

***kWDOG\_PowerOnResetFlag*** Power On flag, set when reset is the result of a powerOnReset.

***kWDOG\_TimeoutResetFlag*** Timeout flag, set when reset is the result of a timeout.

***kWDOG\_SoftwareResetFlag*** Software flag, set when reset is the result of a software.

***kWDOG\_InterruptFlag*** interrupt flag, whether interrupt has occurred or not

## 66.5 Function Documentation

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

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

```
* wdogConfig->enableWdog = true;
* wdogConfig->workMode.enableWait = true;
* wdogConfig->workMode.enableStop = false;
```



```

* wdogConfig->workMode.enableDebug = false;
* wdogConfig->enableInterrupt = false;
* wdogConfig->enablePowerdown = false;
* wdogConfig->resetExtension = false;
* wdogConfig->timeoutValue = 0xFFU;
* wdogConfig->interruptTimeValue = 0x04u;
*

```

#### Parameters

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

#### See Also

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

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

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

This is an example.

```

* wdog_config_t config;
* WDOG_GetDefaultConfig(&config);
* config.timeoutValue = 0xffU;
* config->interruptTimeValue = 0x04u;
* WDOG_Init(wdog_base, &config);
*

```

#### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | WDOG peripheral base address |
| <i>config</i> | The configuration of WDOG    |

### 66.5.3 void WDOG\_Deinit ( WDOG\_Type \* *base* )

This function shuts down the WDOG. Watchdog Enable bit is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. This bit(WDE) can be set/reset only in debug mode(exception).

### 66.5.4 static void WDOG\_Enable ( WDOG\_Type \* *base* ) [inline], [static]

This function writes a value into the WDOG\_WCR register to enable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception.

## Parameters

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

**66.5.5 static void WDOG\_Disable ( WDOG\_Type \* *base* ) [inline], [static]**

This function writes a value into the WDOG\_WCR register to disable the WDOG. This is a write one once only bit. It is not possible to clear this bit by a software write, once the bit is set. only debug mode exception

## Parameters

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

**66.5.6 static void WDOG\_TriggerSystemSoftwareReset ( WDOG\_Type \* *base* ) [inline], [static]**

This function will write to the WCR[SRS] bit to trigger a software system reset. This bit will automatically resets to "1" after it has been asserted to "0". Note: Calling this API will reset the system right now, please using it with more attention.

## Parameters

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

**66.5.7 static void WDOG\_TriggerSoftwareSignal ( WDOG\_Type \* *base* ) [inline], [static]**

This function will write to the WCR[WDA] bit to trigger WDOG\_B signal assertion. The WDOG\_B signal can be routed to external pin of the chip, the output pin will turn to assertion along with WDOG\_B signal. Note: The WDOG\_B signal will remain assert until a power on reset occurred, so, please take more attention while calling it.

## Parameters

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

### 66.5.8 static void WDOG\_EnableInterrupts ( WDOG\_Type \* *base*, uint16\_t *mask* ) [inline], [static]

This bit is a write once only bit. Once the software does a write access to this bit, it will get locked and cannot be reprogrammed until the next system reset assertion

## Parameters

|             |                                                                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | WDOG peripheral base address                                                                                                                                            |
| <i>mask</i> | The interrupts to enable The parameter can be combination of the following source if defined. <ul style="list-style-type: none"> <li>• kWDOG_InterruptEnable</li> </ul> |

### 66.5.9 uint16\_t WDOG\_GetStatusFlags ( WDOG\_Type \* *base* )

This function gets all reset status flags.

```
* uint16_t status;
* status = WDOG_GetStatusFlags (wdog_base);
*
```

## Parameters

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

## Returns

State of the status flag: asserted (true) or not-asserted (false).

## See Also

[\\_wdog\\_status\\_flags](#)

- true: a related status flag has been set.
- false: a related status flag is not set.

### 66.5.10 void WDOG\_ClearInterruptStatus ( WDOG\_Type \* *base*, uint16\_t *mask* )

This function clears the WDOG status flag.

This is an example for clearing the interrupt flag.

```
* WDOG_ClearStatusFlags (wdog_base, kWDOG_InterruptFlag);
*
```

## Parameters

|             |                                                                                                              |
|-------------|--------------------------------------------------------------------------------------------------------------|
| <i>base</i> | WDOG peripheral base address                                                                                 |
| <i>mask</i> | The status flags to clear. The parameter could be any combination of the following values. kWDOG_TimeoutFlag |

### 66.5.11 static void WDOG\_SetTimeoutValue ( WDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]

This function sets the timeout value. This function writes a value into WCR registers. The time-out value can be written at any point of time but it is loaded to the counter at the time when WDOG is enabled or after the service routine has been performed.

## Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | WDOG peripheral base address                  |
| <i>timeoutCount</i> | WDOG timeout value; count of WDOG clock tick. |

### 66.5.12 static void WDOG\_SetInterruptTimeoutValue ( WDOG\_Type \* *base*, uint16\_t *timeoutCount* ) [inline], [static]

This function sets the interrupt count timeout value. This function writes a value into WIC registers which are write-once. This field is write once only. Once the software does a write access to this field, it will get locked and cannot be reprogrammed until the next system reset assertion.

## Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | WDOG peripheral base address                  |
| <i>timeoutCount</i> | WDOG timeout value; count of WDOG clock tick. |

### 66.5.13 static void WDOG\_DisablePowerDownEnable ( WDOG\_Type \* *base* ) [inline], [static]

This function disable the WDOG power down enable(PDE). This function writes a value into WMCR registers which are write-once. This field is write once only. Once software sets this bit it cannot be reset until the next system reset.

## Parameters

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

**66.5.14 void WDOG\_Refresh ( WDOG\_Type \* *base* )**

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

## Parameters

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

## Chapter 67

# XECC: external error correction code controller

### 67.1 Overview

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

The XECC module supports Single Error Correction and Double Error Detection (SEC-DED) ECC function to provide reliability for external memory ECC region access.

This example code shows how to correct single error and detect multiple error using the XECC driver.

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

### Data Structures

- struct `xecc_config_t`  
*XECC user configuration. [More...](#)*
- struct `xecc_single_error_info_t`  
*XECC single error information, including single error address, ECC code, error data, error bit position and error bit field. [More...](#)*
- struct `xecc_multi_error_info_t`  
*XECC multiple error information, including multiple error address, ECC code, error data and error bit field. [More...](#)*

### Enumerations

- enum {  
    `kXECC_SingleErrorInterruptEnable` = `XECC_ERR_SIG_EN_SINGLE_ERR_SIG_EN_MASK`,  
    `kXECC_MultiErrorInterruptEnable` = `XECC_ERR_SIG_EN_MULTI_ERR_SIG_EN_MASK`,  
    `kXECC_AllInterruptsEnable` }  
*XECC interrupt configuration structure, , `xecc_interrupt_enable_t`.*
- enum {  
    `kXECC_SingleErrorInterruptStatusEnable`,  
    `kXECC_MultiErrorInterruptStatusEnable`,  
    `kXECC_AllInterruptsStatusEnable` }  
*XECC interrupt status configuration structure, `xecc_interrupt_status_enable_t`.*
- enum {  
    `kXECC_SingleErrorInterruptFlag`,  
    `kXECC_MultiErrorInterruptFlag`,  
    `kXECC_AllInterruptsFlag` }  
*XECC status flags, `xecc_interrupt_status_t`.*

### Driver version

- #define `FSL_XECC_DRIVER_VERSION` (`MAKE_VERSION(2U, 0U, 0U)`)  
*Driver version 2.0.0.*

## Initialization and deinitialization

- void [XECC\\_Init](#) (XECC\_Type \*base, const [xecc\\_config\\_t](#) \*config)  
*XECC module initialization function.*
- void [XECC\\_Deinit](#) (XECC\_Type \*base)  
*Deinitializes the XECC.*
- void [XECC\\_GetDefaultConfig](#) ([xecc\\_config\\_t](#) \*config)  
*Sets the XECC configuration structure to default values.*

## Status

- static uint32\_t [XECC\\_GetStatusFlags](#) (XECC\_Type \*base)  
*Gets XECC status flags.*
- static void [XECC\\_ClearStatusFlags](#) (XECC\_Type \*base, uint32\_t mask)  
*XECC module clear interrupt status.*
- static void [XECC\\_EnableInterruptStatus](#) (XECC\_Type \*base, uint32\_t mask)  
*XECC module enable interrupt status.*
- static void [XECC\\_DisableInterruptStatus](#) (XECC\_Type \*base, uint32\_t mask)  
*XECC module disable interrupt status.*

## Interrupts

- static void [XECC\\_EnableInterrupts](#) (XECC\_Type \*base, uint32\_t mask)  
*XECC module enable interrupt.*
- static void [XECC\\_DisableInterrupts](#) (XECC\_Type \*base, uint32\_t mask)  
*XECC module disable interrupt.*

## functional

- static void [XECC\\_WriteECCEnable](#) (XECC\_Type \*base, bool enable)  
*XECC module write ECC function enable.*
- static void [XECC\\_ReadECCEnable](#) (XECC\_Type \*base, bool enable)  
*XECC module read ECC function enable.*
- static void [XECC\\_SwapECCEnable](#) (XECC\_Type \*base, bool enable)  
*XECC module swap data enable.*
- [status\\_t](#) [XECC\\_ErrorInjection](#) (XECC\_Type \*base, uint32\_t errordata, uint8\_t erroreccdata)  
*XECC module error injection.*
- void [XECC\\_GetSingleErrorInfo](#) (XECC\_Type \*base, [xecc\\_single\\_error\\_info\\_t](#) \*info)  
*XECC module get single error information.*
- void [XECC\\_GetMultiErrorInfo](#) (XECC\_Type \*base, [xecc\\_multi\\_error\\_info\\_t](#) \*info)  
*XECC module get multiple error information.*

## 67.2 Data Structure Documentation

### 67.2.1 struct xecc\_config\_t

#### Data Fields

- bool [enableXECC](#)  
*Enable the XECC function.*



- bool [enableWriteECC](#)  
*Enable write ECC function.*
- bool [enableReadECC](#)  
*Enable read ECC function.*
- bool [enableSwap](#)  
*Enable swap function.*
- uint32\_t [Region0BaseAddress](#)  
*ECC region 0 base address.*
- uint32\_t [Region0EndAddress](#)  
*ECC region 0 end address.*
- uint32\_t [Region1BaseAddress](#)  
*ECC region 1 base address.*
- uint32\_t [Region1EndAddress](#)  
*ECC region 1 end address.*
- uint32\_t [Region2BaseAddress](#)  
*ECC region 2 base address.*
- uint32\_t [Region2EndAddress](#)  
*ECC region 2 end address.*
- uint32\_t [Region3BaseAddress](#)  
*ECC region 3 base address.*
- uint32\_t [Region3EndAddress](#)  
*ECC region 3 end address.*

#### Field Documentation

- (1) **bool xecc\_config\_t::enableXECC**
- (2) **bool xecc\_config\_t::enableWriteECC**
- (3) **bool xecc\_config\_t::enableReadECC**
- (4) **bool xecc\_config\_t::enableSwap**

The minimum ECC region range is 4k, so the lower 12 bits of this register must be 0.

- (5) **uint32\_t xecc\_config\_t::Region0BaseAddress**
- (6) **uint32\_t xecc\_config\_t::Region0EndAddress**
- (7) **uint32\_t xecc\_config\_t::Region1BaseAddress**
- (8) **uint32\_t xecc\_config\_t::Region1EndAddress**
- (9) **uint32\_t xecc\_config\_t::Region2BaseAddress**
- (10) **uint32\_t xecc\_config\_t::Region2EndAddress**
- (11) **uint32\_t xecc\_config\_t::Region3BaseAddress**
- (12) **uint32\_t xecc\_config\_t::Region3EndAddress**

### 67.2.2 struct xecc\_single\_error\_info\_t

#### Data Fields

- uint32\_t [singleErrorAddress](#)  
*Single error address.*
- uint32\_t [singleErrorData](#)  
*Single error read data.*
- uint32\_t [singleErrorEccCode](#)  
*Single error ECC code.*
- uint32\_t [singleErrorBitPos](#)  
*Single error bit position.*
- uint32\_t [singleErrorBitField](#)  
*Single error bit field.*

### 67.2.3 struct xecc\_multi\_error\_info\_t

#### Data Fields

- uint32\_t [multiErrorAddress](#)  
*Multiple error address.*
- uint32\_t [multiErrorData](#)  
*Multiple error read data.*
- uint32\_t [multiErrorEccCode](#)  
*Multiple error ECC code.*
- uint32\_t [multiErrorBitField](#)  
*Single error bit field.*

## 67.3 Macro Definition Documentation

### 67.3.1 #define FSL\_XECC\_DRIVER\_VERSION (MAKE\_VERSION(2U, 0U, 0U))

## 67.4 Enumeration Type Documentation

### 67.4.1 anonymous enum

This structure contains the settings for all of the XECC interrupt configurations.

Enumerator

- kXECC\_SingleErrorInterruptEnable*** Single bit error interrupt enable.
- kXECC\_MultiErrorInterruptEnable*** Multiple bit error interrupt enable.
- kXECC\_AllInterruptsEnable*** all interrupts enable

### 67.4.2 anonymous enum

This structure contains the settings for all of the XECC interrupt status configurations.

Enumerator

***kXECC\_SingleErrorInterruptStatusEnable*** Single bit error interrupt status enable.  
***kXECC\_MultiErrorInterruptStatusEnable*** Multiple bits error interrupt status enable.  
***kXECC\_AllInterruptsStatusEnable*** all interrupts enable

### 67.4.3 anonymous enum

This provides constants for the XECC status flags for use in the XECC functions.

Enumerator

***kXECC\_SingleErrorInterruptFlag*** Single bit error interrupt happens on read data.  
***kXECC\_MultiErrorInterruptFlag*** Multiple bits error interrupt happens on read data.  
***kXECC\_AllInterruptsFlag*** all interrupts happens on read data

## 67.5 Function Documentation

### 67.5.1 void XECC\_Init ( XECC\_Type \* *base*, const xecc\_config\_t \* *config* )

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | XECC base address.                           |
| <i>config</i> | pointer to the XECC configuration structure. |

### 67.5.2 void XECC\_Deinit ( XECC\_Type \* *base* )

Parameters

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

### 67.5.3 void XECC\_GetDefaultConfig ( xecc\_config\_t \* *config* )

## Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>config</i> | pointer to the XECC configuration structure. |
|---------------|----------------------------------------------|

#### 67.5.4 static uint32\_t XECC\_GetStatusFlags ( XECC\_Type \* *base* ) [inline], [static]

## Parameters

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

## Returns

XECC status flags.

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

## Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>base</i> | XECC base address.                            |
| <i>mask</i> | status to clear from xecc_interrupt_status_t. |

#### 67.5.6 static void XECC\_EnableInterruptStatus ( XECC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>base</i> | XECC base address.                                    |
| <i>mask</i> | status to enable from xecc_interrupt_status_enable_t. |

#### 67.5.7 static void XECC\_DisableInterruptStatus ( XECC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                      |
|-------------|----------------------------------------------------------------------|
| <i>base</i> | XECC base address.                                                   |
| <i>mask</i> | status to disable from <code>xecc_interrupt_status_enable_t</code> . |

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

## Parameters

|             |                                                                      |
|-------------|----------------------------------------------------------------------|
| <i>base</i> | XECC base address.                                                   |
| <i>mask</i> | The interrupts to enable from <code>xecc_interrupt_enable_t</code> . |

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

## Parameters

|             |                                                                       |
|-------------|-----------------------------------------------------------------------|
| <i>base</i> | XECC base address.                                                    |
| <i>mask</i> | The interrupts to disable from <code>xecc_interrupt_enable_t</code> . |

**67.5.10 static void XECC\_WriteECCEnable ( XECC\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

## Parameters

|               |                    |
|---------------|--------------------|
| <i>base</i>   | XECC base address. |
| <i>enable</i> | enable or disable. |

**67.5.11 static void XECC\_ReadECCEnable ( XECC\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

## Parameters

|               |                    |
|---------------|--------------------|
| <i>base</i>   | XECC base address. |
| <i>enable</i> | enable or disable. |

**67.5.12** `static void XECC_SwapECCEnable ( XECC_Type * base, bool enable )`  
**[inline], [static]**

## Parameters

|               |                    |
|---------------|--------------------|
| <i>base</i>   | XECC base address. |
| <i>enable</i> | enable or disable. |

**67.5.13** `status_t XECC_ErrorInjection ( XECC_Type * base, uint32_t errordata,  
uint8_t erroreccdata )`

## Parameters

|                     |                    |
|---------------------|--------------------|
| <i>base</i>         | XECC base address. |
| <i>errordata</i>    | error data.        |
| <i>erroreccdata</i> | ecc code.          |

## Return values

|                         |  |
|-------------------------|--|
| <i>kStatus_Success.</i> |  |
|-------------------------|--|

**67.5.14** `void XECC_GetSingleErrorInfo ( XECC_Type * base,  
xecc_single_error_info_t * info )`

## Parameters

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

|             |                           |
|-------------|---------------------------|
| <i>info</i> | single error information. |
|-------------|---------------------------|

**67.5.15** void XECC\_GetMultiErrorInfo ( XECC\_Type \* *base*, xecc\_multi\_error\_info\_t \* *info* )

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | XECC base address.          |
| <i>info</i> | multiple error information. |

## Chapter 68

# XBARA: Inter-Peripheral Crossbar Switch

### 68.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARA) block of MCUXpresso SDK devices.

The XBARA peripheral driver configures the XBARA (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARA module.

XBARA driver has two parts:

- Signal connection interconnects input and output signals.
- Active edge feature - Some of the outputs provide an active edge detection. If an active edge occurs, an interrupt or a DMA request can be called. APIs handle user callbacks for the interrupts. The driver also includes API for clearing and reading status bits.

### 68.2 Function

#### 68.2.1 XBARA Initialization

To initialize the XBARA driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARA module is initialized by calling the [XBARA\\_Init\(\)](#) function.

#### 68.2.2 Call diagram

1. Call the "XBARA\_Init()" function to initialize the XBARA module.
2. Optionally, call the "XBARA\_SetSignalsConnection()" function to Set connection between the selected XBARA\_IN[\*] input and the XBARA\_OUT[\*] output signal. It connects the XBARA input to the selected XBARA output. A configuration structure of the "xbara\_input\_signal\_t" type and "xbara\_output\_signal\_t" type is required.
3. Call the "XBARA\_SetOutputSignalConfig" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbara\_control\_config\_t" type is required to point to structure that keeps configuration of control register.
4. Finally, the XBARA works properly.

### 68.3 Typical use case

#### Data Structures

- struct [xbara\\_control\\_config\\_t](#)  
*Defines the configuration structure of the XBARA control register. [More...](#)*



## Enumerations

- enum `xbara_active_edge_t` {  
`kXBARA_EdgeNone` = 0U,  
`kXBARA_EdgeRising` = 1U,  
`kXBARA_EdgeFalling` = 2U,  
`kXBARA_EdgeRisingAndFalling` = 3U }  
*XBARA active edge for detection.*
- enum `xbara_request_t` {  
`kXBARA_RequestDisable` = 0U,  
`kXBARA_RequestDMAEnable` = 1U,  
`kXBARA_RequestInterruptEnalbe` = 2U }  
*Defines the XBARA DMA and interrupt configurations.*
- enum `xbara_status_flag_t` {  
`kXBARA_EdgeDetectionOut0`,  
`kXBARA_EdgeDetectionOut1`,  
`kXBARA_EdgeDetectionOut2`,  
`kXBARA_EdgeDetectionOut3` }  
*XBARA status flags.*

## XBARA functional Operation.

- void `XBARA_Init` (XBARA\_Type \*base)  
*Initializes the XBARA module.*
- void `XBARA_Deinit` (XBARA\_Type \*base)  
*Shuts down the XBARA module.*
- void `XBARA_SetSignalsConnection` (XBARA\_Type \*base, xbar\_input\_signal\_t input, xbar\_output\_signal\_t output)  
*Sets a connection between the selected XBARA\_IN[\*] input and the XBARA\_OUT[\*] output signal.*
- uint32\_t `XBARA_GetStatusFlags` (XBARA\_Type \*base)  
*Gets the active edge detection status.*
- void `XBARA_ClearStatusFlags` (XBARA\_Type \*base, uint32\_t mask)  
*Clears the edge detection status flags of relative mask.*
- void `XBARA_SetOutputSignalConfig` (XBARA\_Type \*base, xbar\_output\_signal\_t output, const `xbara_control_config_t` \*controlConfig)  
*Configures the XBARA control register.*

## 68.4 Data Structure Documentation

### 68.4.1 struct `xbara_control_config_t`

This structure keeps the configuration of XBARA control register for one output. Control registers are available only for a few outputs. Not every XBARA module has control registers.

## Data Fields

- `xbara_active_edge_t` `activeEdge`

- *Active edge to be detected.*  
**xbara\_request\_t requestType**  
*Selects DMA/Interrupt request.*

### Field Documentation

(1) **xbara\_active\_edge\_t xbara\_control\_config\_t::activeEdge**

(2) **xbara\_request\_t xbara\_control\_config\_t::requestType**

## 68.5 Enumeration Type Documentation

### 68.5.1 enum xbara\_active\_edge\_t

Enumerator

- kXBARA\_EdgeNone** Edge detection status bit never asserts.
- kXBARA\_EdgeRising** Edge detection status bit asserts on rising edges.
- kXBARA\_EdgeFalling** Edge detection status bit asserts on falling edges.
- kXBARA\_EdgeRisingAndFalling** Edge detection status bit asserts on rising and falling edges.

### 68.5.2 enum xbara\_request\_t

Enumerator

- kXBARA\_RequestDisable** Interrupt and DMA are disabled.
- kXBARA\_RequestDMAEnable** DMA enabled, interrupt disabled.
- kXBARA\_RequestInterruptEnable** Interrupt enabled, DMA disabled.

### 68.5.3 enum xbara\_status\_flag\_t

This provides constants for the XBARA status flags for use in the XBARA functions.

Enumerator

- kXBARA\_EdgeDetectionOut0** XBAR\_OUT0 active edge interrupt flag, sets when active edge detected.
- kXBARA\_EdgeDetectionOut1** XBAR\_OUT1 active edge interrupt flag, sets when active edge detected.
- kXBARA\_EdgeDetectionOut2** XBAR\_OUT2 active edge interrupt flag, sets when active edge detected.
- kXBARA\_EdgeDetectionOut3** XBAR\_OUT3 active edge interrupt flag, sets when active edge detected.

## 68.6 Function Documentation

### 68.6.1 void XBARA\_Init ( XBARA\_Type \* *base* )

This function un-gates the XBARA clock.

## Parameters

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

**68.6.2 void XBARA\_Deinit ( XBARA\_Type \* *base* )**

This function disables XBARA clock.

## Parameters

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

**68.6.3 void XBARA\_SetSignalsConnection ( XBARA\_Type \* *base*,  
xbar\_input\_signal\_t *input*, xbar\_output\_signal\_t *output* )**

This function connects the XBARA input to the selected XBARA output. If more than one XBARA module is available, only the inputs and outputs from the same module can be connected.

Example:

```
XBARA_SetSignalsConnection(XBARA, kXBARA_InputPIT_TRG0, kXBARA_OutputDMAMUX18);
```

## Parameters

|               |                           |
|---------------|---------------------------|
| <i>base</i>   | XBARA peripheral address. |
| <i>input</i>  | XBARA input signal.       |
| <i>output</i> | XBARA output signal.      |

**68.6.4 uint32\_t XBARA\_GetStatusFlags ( XBARA\_Type \* *base* )**

This function gets the active edge detect status of all XBARA\_OUTs. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBARA\_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>base</i> | XBARA peripheral address. |
|-------------|---------------------------|

## Returns

the mask of these status flag bits.

### 68.6.5 void XBARA\_ClearStatusFlags ( XBARA\_Type \* *base*, uint32\_t *mask* )

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | XBARA peripheral address.  |
| <i>mask</i> | the status flags to clear. |

### 68.6.6 void XBARA\_SetOutputSignalConfig ( XBARA\_Type \* *base*, xbar\_output\_signal\_t *output*, const xbara\_control\_config\_t \* *controlConfig* )

This function configures an XBARA control register. The active edge detection and the DMA/IRQ function on the corresponding XBARA output can be set.

Example:

```
xbara_control_config_t userConfig;
userConfig.activeEdge = kXBARA_EdgeRising;
userConfig.requestType = kXBARA_RequestInterruptEnalbe;
XBARA_SetOutputSignalConfig(XBARA, kXBARA_OutputDMAMUX18, &userConfig);
```

## Parameters

|                      |                                                                    |
|----------------------|--------------------------------------------------------------------|
| <i>base</i>          | XBARA peripheral address.                                          |
| <i>output</i>        | XBARA output number.                                               |
| <i>controlConfig</i> | Pointer to structure that keeps configuration of control register. |

## Chapter 69

# XBARB: Inter-Peripheral Crossbar Switch

### 69.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Peripheral Crossbar Switch (XBARB) block of MCUXpresso SDK devices.

The XBARB peripheral driver configures the XBARB (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBARB module.

XBARB driver has two parts:

- Signal connection interconnects input and output signals.

### 69.2 Function groups

#### 69.2.1 XBARB Initialization

To initialize the XBARB driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBARB module is initialized by calling the [XBARB\\_Init\(\)](#) function.

#### 69.2.2 Call diagram

1. Call the "XBARB\_Init()" function to initialize the XBARB module.
2. Optionally, call the "XBARB\_SetSignalsConnection()" function to Set connection between the selected XBARB\_IN[\*] input and the XBARB\_OUT[\*] output signal. It connects the XBARB input to the selected XBARB output. A configuration structure of the "xbarb\_input\_signal\_t" type and "xbarb\_output\_signal\_t" type is required.
3. Finally, the XBARB works properly.

### 69.3 Typical use case

#### XBARB functional Operation.

- void [XBARB\\_Init](#) (XBARB\_Type \*base)  
*Initializes the XBARB module.*
- void [XBARB\\_Deinit](#) (XBARB\_Type \*base)  
*Shuts down the XBARB module.*
- void [XBARB\\_SetSignalsConnection](#) (XBARB\_Type \*base, xbarb\_input\_signal\_t input, xbarb\_output\_signal\_t output)  
*Configures a connection between the selected XBARB\_IN[\*] input and the XBARB\_OUT[\*] output signal.*

## 69.4 Function Documentation

### 69.4.1 void XBARB\_Init ( XBARB\_Type \* *base* )

This function un-gates the XBARB clock.

## Parameters

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

**69.4.2 void XBARB\_Deinit ( XBARB\_Type \* *base* )**

This function disables XBARB clock.

## Parameters

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

**69.4.3 void XBARB\_SetSignalsConnection ( XBARB\_Type \* *base*,  
xbar\_input\_signal\_t *input*, xbar\_output\_signal\_t *output* )**

This function configures which XBARB input is connected to the selected XBARB output. If more than one XBARB module is available, only the inputs and outputs from the same module can be connected.

## Parameters

|               |                           |
|---------------|---------------------------|
| <i>base</i>   | XBARB peripheral address. |
| <i>input</i>  | XBARB input signal.       |
| <i>output</i> | XBARB output signal.      |



## Chapter 70

# XRDC2: Extended Resource Domain Controller 2

### 70.1 Overview

The MCUXpresso SDK provides a driver for the Extended Resource Domain Controller 2 (XRDC2) block of MCUXpresso SDK devices.

### 70.2 XRDC2 functions

XRDC2 module includes such submodules:

- XRDC2\_MGR The Manager submodule coordinates all programming model reads and writes.
- XRDC2\_MDAC The Master Domain Assignment Controller handles resource assignments and generation of the domain identifiers (domain ID).
- XRDC2\_MSC The Memory Slot Controller implements the access controls for slave memory slots based on the pre-programmed region descriptor registers.
- XRDC2\_MRC The Memory Region Controller implements the access controls for slave memories based on the pre-programmed region descriptor registers.
- XRDC2\_PAC The Peripheral Access Controller implements the access controls for slave peripherals based on the preprogrammed domain access control registers.

Accordingly, the XRDC2 driver functions could be grouped as follows:

- XRDC2\_MGR functions.
- XRDC2\_MDAC functions.
- XRDC2\_MSC functions.
- XRDC2\_MRC functions.
- XRDC2\_PAC functions.

### 70.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/xrdc2

### Data Structures

- struct [xrdc2\\_master\\_domain\\_assignment\\_t](#)  
*Domain assignment for the bus master. [More...](#)*
- struct [xrdc2\\_periph\\_access\\_config\\_t](#)  
*XRDC2 peripheral domain access control configuration. [More...](#)*
- struct [xrdc2\\_mem\\_access\\_config\\_t](#)  
*XRDC2 memory region domain access control configuration. [More...](#)*
- struct [xrdc2\\_mem\\_slot\\_access\\_config\\_t](#)  
*XRDC2 memory slot domain access control configuration. [More...](#)*

## Macros

- #define **FSL\_XRDC2\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 2))  
*Driver version.*

## Enumerations

- enum **xrdc2\_global\_config\_lock\_t** {  
  **kXRDC2\_GlobalConfigLockDisabled**,  
  **kXRDC2\_GlobalConfigLockDisabledUntilReset**,  
  **kXRDC2\_GlobalConfigLockOwnerOnly**,  
  **kXRDC2\_GlobalConfigLockEnabledUntilReset** }  
  *Global configuration lock.*
- enum **xrdc2\_secure\_attr\_t** {  
  **kXRDC2\_MasterSecure** = 0,  
  **kXRDC2\_ForceSecure** = 2,  
  **kXRDC2\_ForceNonSecure** = 3 }  
  *XRDC2 secure attribute, the register bit MDACi\_MDAj\_W0[SA], secure/nonsecure attribute output on a hit.*
- enum **xrdc2\_privilege\_attr\_t** {  
  **kXRDC2\_MasterPrivilege** = 0,  
  **kXRDC2\_ForceUser** = 2,  
  **kXRDC2\_ForcePrivilege** = 3 }  
  *XRDC2 privileged attribute, the register bit MDACi\_MDAj\_W0[PA], defines the privileged/user attribute on a hit.*
- enum **xrdc2\_access\_policy\_t**  
  *XRDC2 domain access control policy.*
- enum **xrdc2\_access\_config\_lock\_t** {  
  **kXRDC2\_AccessConfigLockDisabled** = 0U,  
  **kXRDC2\_AccessConfigLockDisabledUntilReset** = 1U,  
  **kXRDC2\_AccessConfigLockDomainXOnly** = 2U,  
  **kXRDC2\_AccessConfigLockEnabledUntilReset** = 3U }  
  *Access configuration lock mode, the register field PDAC and MRGD LK2.*

## Functions

- void **XRDC2\_Init** (XRDC2\_Type \*base)  
  *Initializes the XRDC2 module.*
- void **XRDC2\_Deinit** (XRDC2\_Type \*base)  
  *De-initializes the XRDC2 module.*

## XRDC2 manager (XRDC2)

- void **XRDC2\_SetGlobalValid** (XRDC2\_Type \*base, bool valid)  
  *Sets the XRDC2 global valid.*
- static uint8\_t **XRDC2\_GetCurrentMasterDomainId** (XRDC2\_Type \*base)  
  *Gets the domain ID of the current bus master.*
- static void **XRDC2\_SetGlobalConfigLock** (XRDC2\_Type \*base, **xrdc2\_global\_config\_lock\_t** mode)

- *Set the global configuration lock mode.*
- static uint8\_t [XRDC2\\_GetCurrentGlobalConfigLockOwnerDomainId](#) (XRDC2\_Type \*base)  
*Gets the domain ID of global configuration lock owner.*

## XRDC2 Master Domain Assignment Controller (XRDC2\_MDAC).

- void [XRDC2\\_GetDefaultMasterDomainAssignment](#) (xrdc2\_master\_domain\_assignment\_t \*assignment)  
*Gets the default master domain assignment.*
- void [XRDC2\\_SetMasterDomainAssignment](#) (XRDC2\_Type \*base, xrdc2\_master\_t master, uint8\_t assignIndex, const xrdc2\_master\_domain\_assignment\_t \*assignment)  
*Sets the processor bus master domain assignment.*
- static void [XRDC2\\_LockMasterDomainAssignment](#) (XRDC2\_Type \*base, xrdc2\_master\_t master, uint8\_t assignIndex)  
*Locks the bus master domain assignment register.*
- static void [XRDC2\\_SetMasterDomainAssignmentValid](#) (XRDC2\_Type \*base, xrdc2\_master\_t master, uint8\_t assignIndex, bool valid)  
*Sets the master domain assignment as valid or invalid.*

## XRDC2 Memory Slot Access Controller (XRDC2\_MSC).

- void [XRDC2\\_GetMemSlotAccessDefaultConfig](#) (xrdc2\_mem\_slot\_access\_config\_t \*config)  
*Gets the default memory slot access configuration.*
- void [XRDC2\\_SetMemSlotAccessConfig](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot, const xrdc2\_mem\_slot\_access\_config\_t \*config)  
*Sets the memory slot access policy.*
- void [XRDC2\\_SetMemSlotAccessValid](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot, bool valid)  
*Sets the memory slot descriptor as valid or invalid.*
- void [XRDC2\\_SetMemSlotAccessLockMode](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot, xrdc2\_access\_config\_lock\_t lockMode)  
*Sets the memory slot descriptor lock mode.*
- void [XRDC2\\_SetMemSlotDomainAccessPolicy](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot, uint8\_t domainId, xrdc2\_access\_policy\_t policy)  
*Sets the memory slot access policy for specific domain.*
- void [XRDC2\\_EnableMemSlotExclAccessLock](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot, bool enable)  
*Enable or disable the memory slot exclusive access lock.*
- uint8\_t [XRDC2\\_GetMemSlotExclAccessLockDomainOwner](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot)  
*Get current memory slot exclusive access lock owner.*
- status\_t [XRDC2\\_TryLockMemSlotExclAccess](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot)  
*Try to lock the memory slot exclusive access.*
- void [XRDC2\\_LockMemSlotExclAccess](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot)  
*Lock the memory slot exclusive access using blocking method.*
- static void [XRDC2\\_UnlockMemSlotExclAccess](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot)  
*Unlock the memory slot exclusive access.*
- static void [XRDC2\\_ForceMemSlotExclAccessLockRelease](#) (XRDC2\_Type \*base, xrdc2\_mem\_slot\_t memSlot)  
*Force the memory slot exclusive access lock release.*

## XRDC2 Memory Region Controller (XRDC2\_MRC)

- void [XRDC2\\_GetMemAccessDefaultConfig](#) (xrdc2\_mem\_access\_config\_t \*config)  
*Gets the default memory access configuration.*
- void [XRDC2\\_SetMemAccessConfig](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem, const xrdc2\_mem\_access\_config\_t \*config)  
*Sets the memory region access policy.*
- void [XRDC2\\_SetMemAccessValid](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem, bool valid)  
*Sets the memory region descriptor as valid or invalid.*
- void [XRDC2\\_SetMemAccessLockMode](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem, xrdc2\_access\_config\_lock\_t lockMode)  
*Sets the memory descriptor lock mode.*
- void [XRDC2\\_SetMemDomainAccessPolicy](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem, uint8\_t domainId, xrdc2\_access\_policy\_t policy)  
*Sets the memory region access policy for specific domain.*
- void [XRDC2\\_EnableMemExclAccessLock](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem, bool enable)  
*Enable or disable the memory region exclusive access lock.*
- uint8\_t [XRDC2\\_GetMemExclAccessLockDomainOwner](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem)  
*Get current memory region exclusive access lock owner.*
- status\_t [XRDC2\\_TryLockMemExclAccess](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem)  
*Try to lock the memory region exclusive access.*
- void [XRDC2\\_LockMemExclAccess](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem)  
*Lock the memory region exclusive access using blocking method.*
- void [XRDC2\\_UnlockMemExclAccess](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem)  
*Unlock the memory region exclusive access.*
- void [XRDC2\\_ForceMemExclAccessLockRelease](#) (XRDC2\_Type \*base, xrdc2\_mem\_t mem)  
*Force the memory region exclusive access lock release.*

## XRDC2 Peripheral Access Controller (XRDC2\_PAC)

- void [XRDC2\\_GetPeriphAccessDefaultConfig](#) (xrdc2\_periph\_access\_config\_t \*config)  
*Gets the default peripheral access configuration.*
- void [XRDC2\\_SetPeriphAccessConfig](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph, const xrdc2\_periph\_access\_config\_t \*config)  
*Sets the peripheral access policy.*
- void [XRDC2\\_SetPeriphAccessValid](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph, bool valid)  
*Sets the peripheral descriptor as valid or invalid.*
- void [XRDC2\\_SetPeriphAccessLockMode](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph, xrdc2\_access\_config\_lock\_t lockMode)  
*Sets the peripheral descriptor lock mode.*
- void [XRDC2\\_SetPeriphDomainAccessPolicy](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph, uint8\_t domainId, xrdc2\_access\_policy\_t policy)  
*Sets the peripheral access policy for specific domain.*
- void [XRDC2\\_EnablePeriphExclAccessLock](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph, bool enable)  
*Disable the peripheral exclusive access lock.*
- uint8\_t [XRDC2\\_GetPeriphExclAccessLockDomainOwner](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph)  
*Get current peripheral exclusive access lock owner.*

- [status\\_t XRDC2\\_TryLockPeriphExclAccess](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph)  
*Try to lock the peripheral exclusive access.*
- void [XRDC2\\_LockPeriphExclAccess](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph)  
*Lock the peripheral exclusive access using blocking method.*
- void [XRDC2\\_UnlockPeriphExclAccess](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph)  
*Unlock the peripheral exclusive access.*
- void [XRDC2\\_ForcePeriphExclAccessLockRelease](#) (XRDC2\_Type \*base, xrdc2\_periph\_t periph)  
*Force the peripheral exclusive access lock release.*

## 70.4 Data Structure Documentation

### 70.4.1 struct xrdc2\_master\_domain\_assignment\_t

XRDC2 compares the bus master *match input* with the parameter [mask](#) and [match](#) in this structure. If hit, the domain ID, privilege attribute, and secure attribute are used for the access.

#### Data Fields

- bool [lock](#)  
*Set true to lock the descriptor.*
- [xrdc2\\_privilege\\_attr\\_t privilegeAttr](#)  
*Privilege attribute.*
- [xrdc2\\_secure\\_attr\\_t secureAttr](#)  
*Secure attribute.*
- uint8\_t [domainId](#)  
*Domain ID used when this descriptor hit.*
- uint16\_t [mask](#)  
*Mask used for descriptor hit.*
- uint16\_t [match](#)  
*Match used for descriptor hit.*

#### Field Documentation

- (1) bool xrdc2\_master\_domain\_assignment\_t::lock
- (2) xrdc2\_privilege\_attr\_t xrdc2\_master\_domain\_assignment\_t::privilegeAttr
- (3) xrdc2\_secure\_attr\_t xrdc2\_master\_domain\_assignment\_t::secureAttr
- (4) uint8\_t xrdc2\_master\_domain\_assignment\_t::domainId
- (5) uint16\_t xrdc2\_master\_domain\_assignment\_t::mask
- (6) uint16\_t xrdc2\_master\_domain\_assignment\_t::match

### 70.4.2 struct xrdc2\_periph\_access\_config\_t

#### Data Fields

- [xrdc2\\_access\\_config\\_lock\\_t lockMode](#)  
*PDACn lock configuration.*
- [xrdc2\\_access\\_policy\\_t policy](#) [FSL\_FEATURE\_XRDC2\_DOMAIN\_COUNT]  
*Access policy for each domain.*

#### Field Documentation

- (1) [xrdc2\\_access\\_config\\_lock\\_t xrdc2\\_periph\\_access\\_config\\_t::lockMode](#)
- (2) [xrdc2\\_access\\_policy\\_t xrdc2\\_periph\\_access\\_config\\_t::policy](#)[FSL\_FEATURE\_XRDC2\_DOMAIN\_COUNT]

### 70.4.3 struct xrdc2\_mem\_access\_config\_t

#### Data Fields

- [uint32\\_t startAddr](#)  
*Memory region start address, should be 4k aligned.*
- [uint32\\_t endAddr](#)  
*Memory region end address, (endAddr + 1) should be 4k aligned.*
- [xrdc2\\_access\\_config\\_lock\\_t lockMode](#)  
*MRGDn lock configuration.*
- [xrdc2\\_access\\_policy\\_t policy](#) [FSL\_FEATURE\_XRDC2\_DOMAIN\_COUNT]  
*Access policy for each domain.*

#### Field Documentation

- (1) [uint32\\_t xrdc2\\_mem\\_access\\_config\\_t::startAddr](#)
- (2) [uint32\\_t xrdc2\\_mem\\_access\\_config\\_t::endAddr](#)
- (3) [xrdc2\\_access\\_config\\_lock\\_t xrdc2\\_mem\\_access\\_config\\_t::lockMode](#)
- (4) [xrdc2\\_access\\_policy\\_t xrdc2\\_mem\\_access\\_config\\_t::policy](#)[FSL\_FEATURE\_XRDC2\_DOMAIN\_COUNT]

### 70.4.4 struct xrdc2\_mem\_slot\_access\_config\_t

#### Data Fields

- [xrdc2\\_access\\_config\\_lock\\_t lockMode](#)  
*Descriptor lock configuration.*
- [xrdc2\\_access\\_policy\\_t policy](#) [FSL\_FEATURE\_XRDC2\_DOMAIN\_COUNT]  
*Access policy for each domain.*

## Field Documentation

- (1) `xrdc2_access_config_lock_t xrdc2_mem_slot_access_config_t::lockMode`
- (2) `xrdc2_access_policy_t xrdc2_mem_slot_access_config_t::policy[FSL_FEATURE_XRDC2_DOMAIN_COUNT]`

## 70.5 Macro Definition Documentation

70.5.1 `#define FSL_XRDC2_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 70.6 Enumeration Type Documentation

70.6.1 `enum xrdc2_global_config_lock_t`

## Enumerator

*kXRDC2\_GlobalConfigLockDisabled* Lock disabled, registers can be written by any domain.

*kXRDC2\_GlobalConfigLockDisabledUntilReset* Lock disabled until the next reset.

*kXRDC2\_GlobalConfigLockOwnerOnly* Lock enabled, only the lock owner can write.

*kXRDC2\_GlobalConfigLockEnabledUntilReset* Lock enabled, all registers are read only until the next reset.

70.6.2 `enum xrdc2_secure_attr_t`

## Enumerator

*kXRDC2\_MasterSecure* Use the bus master's secure/nonsecure attribute directly.

*kXRDC2\_ForceSecure* Force the bus attribute for this master to secure.

*kXRDC2\_ForceNonSecure* Force the bus attribute for this master to non-secure.

70.6.3 `enum xrdc2_privilege_attr_t`

## Enumerator

*kXRDC2\_MasterPrivilege* Use the bus master's attribute directly.

*kXRDC2\_ForceUser* Force the bus attribute for this master to user.

*kXRDC2\_ForcePrivilege* Force the bus attribute for this master to privileged.

70.6.4 `enum xrdc2_access_config_lock_t`

## Enumerator

*kXRDC2\_AccessConfigLockDisabled* Entire PDACn/MRGDn/MSD can be written.

***kXRDC2\_AccessConfigLockDisabledUntilReset*** Entire PDACn/MRGDn/MSn can be written until next reset.

***kXRDC2\_AccessConfigLockDomainXOnly*** Domain x only write the DxACP field.

***kXRDC2\_AccessConfigLockEnabledUntilReset*** PDACn/MRGDn/MSn is read-only until the next reset.

## 70.7 Function Documentation

### 70.7.1 void XRDC2\_Init ( XRDC2\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
|-------------|--------------------------------|

### 70.7.2 void XRDC2\_Deinit ( XRDC2\_Type \* *base* )

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
|-------------|--------------------------------|

### 70.7.3 void XRDC2\_SetGlobalValid ( XRDC2\_Type \* *base*, bool *valid* )

This function sets the XRDC2 global valid or invalid. When the XRDC2 is global invalid, all accesses from all bus masters to all slaves are allowed.

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | XRDC2 peripheral base address. |
| <i>valid</i> | True to valid XRDC2.           |

### 70.7.4 static uint8\_t XRDC2\_GetCurrentMasterDomainId ( XRDC2\_Type \* *base* ) [inline], [static]

This function returns the domain ID of the current bus master.



## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
|-------------|--------------------------------|

## Returns

Domain ID of current bus master.

### 70.7.5 static void XRDC2\_SetGlobalConfigLock ( XRDC2\_Type \* *base*, xrdc2\_global\_config\_lock\_t *mode* ) [inline], [static]

Once change the lock mode, it could not be changed until next reset.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
| <i>mode</i> | The lock mode.                 |

### 70.7.6 static uint8\_t XRDC2\_GetCurrentGlobalConfigLockOwnerDomainId ( XRDC2\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
|-------------|--------------------------------|

## Returns

Domain ID of the global configuration lock owner.

### 70.7.7 void XRDC2\_GetDefaultMasterDomainAssignment ( xrdc2\_master\_domain\_assignment\_t \* *assignment* )

This function sets the assignment as follows:

```
* config->lock = false;
* config->privilegeAttr = kXRDC2_MasterPrivilege;
* config->secureAttr = kXRDC2_MasterSecure;
* config->domainId = 0U;
* config->mask = 0U;
* config->match = 0U;
*
```

## Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>assignment</i> | Pointer to the assignment structure. |
|-------------------|--------------------------------------|

**70.7.8 void XRDC2\_SetMasterDomainAssignment ( XRDC2\_Type \* *base*, xrdc2\_master\_t *master*, uint8\_t *assignIndex*, const xrdc2\_master\_domain\_assignment\_t \* *assignment* )**

## Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | XRDC2 peripheral base address.       |
| <i>master</i>      | Which master to configure.           |
| <i>assignIndex</i> | Which assignment register to set.    |
| <i>assignment</i>  | Pointer to the assignment structure. |

**70.7.9 static void XRDC2\_LockMasterDomainAssignment ( XRDC2\_Type \* *base*, xrdc2\_master\_t *master*, uint8\_t *assignIndex* ) [inline], [static]**

This function locks the master domain assignment. One bus master might have multiple domain assignment registers. The parameter *assignIndex* specifies which assignment register to lock. After it is locked, the register can't be changed until next reset.

## Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>base</i>        | XRDC2 peripheral base address.     |
| <i>master</i>      | Which master to configure.         |
| <i>assignIndex</i> | Which assignment register to lock. |

**70.7.10 static void XRDC2\_SetMasterDomainAssignmentValid ( XRDC2\_Type \* *base*, xrdc2\_master\_t *master*, uint8\_t *assignIndex*, bool *valid* ) [inline], [static]**

This function sets the master domain assignment as valid or invalid. One bus master might have multiple domain assignment registers. The parameter *assignIndex* specifies which assignment register to configure.

## Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | XRDC2 peripheral base address.            |
| <i>master</i>      | Which master to configure.                |
| <i>assignIndex</i> | Index for the domain assignment register. |
| <i>valid</i>       | True to set valid, false to set invalid.  |

### 70.7.11 void XRDC2\_GetMemSlotAccessDefaultConfig ( xrdc2\_mem\_slot\_access\_config\_t \* *config* )

This function sets the assignment as follows:

```
* config->lockMode = kXRDC2_AccessConfigLockDisabled;
* config->policy[0] = kXRDC2_AccessPolicyNone;
* config->policy[1] = kXRDC2_AccessPolicyNone;
* ...
*
```

## Parameters

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

### 70.7.12 void XRDC2\_SetMemSlotAccessConfig ( XRDC2\_Type \* *base*, xrdc2\_mem\_slot\_t *memSlot*, const xrdc2\_mem\_slot\_access\_config\_t \* *config* )

## Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | XRDC2 peripheral base address.                        |
| <i>memSlot</i> | Which memory slot descriptor to set.                  |
| <i>config</i>  | Pointer to the access policy configuration structure. |

### 70.7.13 void XRDC2\_SetMemSlotAccessValid ( XRDC2\_Type \* *base*, xrdc2\_mem\_slot\_t *memSlot*, bool *valid* )

## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | XRDC2 peripheral base address.           |
| <i>memSlot</i> | Which memory slot descriptor to set.     |
| <i>valid</i>   | True to set valid, false to set invalid. |

**70.7.14** `void XRDC2_SetMemSlotAccessLockMode ( XRDC2_Type * base,  
xrdc2_mem_slot_t memSlot, xrdc2_access_config_lock_t lockMode )`

## Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>base</i>     | XRDC2 peripheral base address.       |
| <i>memSlot</i>  | Which memory slot descriptor to set. |
| <i>lockMode</i> | The lock mode to set.                |

**70.7.15** `void XRDC2_SetMemSlotDomainAccessPolicy ( XRDC2_Type * base,  
xrdc2_mem_slot_t memSlot, uint8_t domainId, xrdc2_access_policy_t  
policy )`

## Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>base</i>     | XRDC2 peripheral base address.                     |
| <i>memSlot</i>  | The memory slot to operate.                        |
| <i>domainId</i> | The ID of the domain whose policy will be changed. |
| <i>policy</i>   | The access policy to set.                          |

**70.7.16** `void XRDC2_EnableMemSlotExclAccessLock ( XRDC2_Type * base,  
xrdc2_mem_slot_t memSlot, bool enable )`

The lock must be enabled first before use. Once disabled, it could not be enabled until reset.

## Parameters

---

|                |                                   |
|----------------|-----------------------------------|
| <i>base</i>    | XRDC2 peripheral base address.    |
| <i>memSlot</i> | The memory slot to operate.       |
| <i>enable</i>  | True to enable, false to disable. |

#### 70.7.17 `uint8_t XRDC2_GetMemSlotExclAccessLockDomainOwner ( XRDC2_Type * base, xrdc2_mem_slot_t memSlot )`

##### Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | XRDC2 peripheral base address. |
| <i>memSlot</i> | The memory slot to operate.    |

##### Returns

The domain ID of the lock owner.

#### 70.7.18 `status_t XRDC2_TryLockMemSlotExclAccess ( XRDC2_Type * base, xrdc2_mem_slot_t memSlot )`

##### Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | XRDC2 peripheral base address. |
| <i>memSlot</i> | The memory slot to operate.    |

##### Return values

|                        |                      |
|------------------------|----------------------|
| <i>kStatus_Fail</i>    | Failed to lock.      |
| <i>kStatus_Success</i> | Locked succussfully. |

#### 70.7.19 `void XRDC2_LockMemSlotExclAccess ( XRDC2_Type * base, xrdc2_mem_slot_t memSlot )`

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | XRDC2 peripheral base address. |
| <i>memSlot</i> | The memory slot to operate.    |

## Note

This function must be called when the lock is not disabled.

**70.7.20 static void XRDC2\_UnlockMemSlotExclAccess ( XRDC2\_Type \* *base*,  
xrdc2\_mem\_slot\_t *memSlot* ) [inline], [static]**

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | XRDC2 peripheral base address. |
| <i>memSlot</i> | The memory slot to operate.    |

## Note

This function must be called by the lock owner.

**70.7.21 static void XRDC2\_ForceMemSlotExclAccessLockRelease ( XRDC2\_Type \*  
*base*, xrdc2\_mem\_slot\_t *memSlot* ) [inline], [static]**

The master does not own the lock could call this function to force release the lock.

## Parameters

|                |                                |
|----------------|--------------------------------|
| <i>base</i>    | XRDC2 peripheral base address. |
| <i>memSlot</i> | The memory slot to operate.    |

**70.7.22 void XRDC2\_GetMemAccessDefaultConfig ( xrdc2\_mem\_access\_config\_t \*  
*config* )**

This function sets the assignment as follows:

```
* config->startAddr = 0U;
* config->endAddr = 0xFFFFFFFFU;
* config->lockMode = kXRDC2_AccessConfigLockDisabled;
* config->policy[0] = kXRDC2_AccessPolicyNone;
* config->policy[1] = kXRDC2_AccessPolicyNone;
* ...
*
```

## Parameters

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

**70.7.23 void XRDC2\_SetMemAccessConfig ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem*, const xrdc2\_mem\_access\_config\_t \* *config* )**

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | XRDC2 peripheral base address.                        |
| <i>mem</i>    | Which memory region descriptor to set.                |
| <i>config</i> | Pointer to the access policy configuration structure. |

**70.7.24 void XRDC2\_SetMemAccessValid ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem*, bool *valid* )**

## Parameters

|              |                                          |
|--------------|------------------------------------------|
| <i>base</i>  | XRDC2 peripheral base address.           |
| <i>mem</i>   | Which memory region descriptor to set.   |
| <i>valid</i> | True to set valid, false to set invalid. |

**70.7.25 void XRDC2\_SetMemAccessLockMode ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem*, xrdc2\_access\_config\_lock\_t *lockMode* )**

## Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | XRDC2 peripheral base address.  |
| <i>mem</i>      | Which memory descriptor to set. |
| <i>lockMode</i> | The lock mode to set.           |

**70.7.26 void XRDC2\_SetMemDomainAccessPolicy ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem*, uint8\_t *domainId*, xrdc2\_access\_policy\_t *policy* )**

## Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>base</i>     | XRDC2 peripheral base address.                     |
| <i>mem</i>      | The memory region to operate.                      |
| <i>domainId</i> | The ID of the domain whose policy will be changed. |
| <i>policy</i>   | The access policy to set.                          |

### 70.7.27 void XRDC2\_EnableMemExclAccessLock ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem*, bool *enable* )

Once disabled, it could not be enabled until reset.

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | XRDC2 peripheral base address.    |
| <i>mem</i>    | The memory region to operate.     |
| <i>enable</i> | True to enable, false to disable. |

### 70.7.28 uint8\_t XRDC2\_GetMemExclAccessLockDomainOwner ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem* )

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
| <i>mem</i>  | The memory region to operate.  |

## Returns

The domain ID of the lock owner.

### 70.7.29 status\_t XRDC2\_TryLockMemExclAccess ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem* )



## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
| <i>mem</i>  | The memory region to operate.  |

## Return values

|                        |                      |
|------------------------|----------------------|
| <i>kStatus_Fail</i>    | Failed to lock.      |
| <i>kStatus_Success</i> | Locked successfully. |

### 70.7.30 void XRDC2\_LockMemExclAccess ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem* )

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
| <i>mem</i>  | The memory region to operate.  |

## Note

This function must be called when the lock is not disabled.

### 70.7.31 void XRDC2\_UnlockMemExclAccess ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem* )

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
| <i>mem</i>  | The memory region to operate.  |

## Note

This function must be called by the lock owner.

### 70.7.32 void XRDC2\_ForceMemExclAccessLockRelease ( XRDC2\_Type \* *base*, xrdc2\_mem\_t *mem* )

The master does not own the lock could call this function to force release the lock.

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
| <i>mem</i>  | The memory region to operate.  |

### 70.7.33 void XRDC2\_GetPeriphAccessDefaultConfig ( xrdc2\_periph\_access\_config\_t \* *config* )

The default configuration is set as follows:

```
* config->lockMode = kXRDC2_AccessConfigLockWritable;
* config->policy[0] = kXRDC2_AccessPolicyNone;
* config->policy[1] = kXRDC2_AccessPolicyNone;
* ...
* config->policy[15] = kXRDC2_AccessPolicyNone;
*
```

## Parameters

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

### 70.7.34 void XRDC2\_SetPeriphAccessConfig ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph*, const xrdc2\_periph\_access\_config\_t \* *config* )

## Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | XRDC2 peripheral base address.                        |
| <i>periph</i> | Which peripheral descriptor to set.                   |
| <i>config</i> | Pointer to the access policy configuration structure. |

### 70.7.35 void XRDC2\_SetPeriphAccessValid ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph*, bool *valid* )

## Parameters

---

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | XRDC2 peripheral base address.           |
| <i>periph</i> | Which peripheral descriptor to set.      |
| <i>valid</i>  | True to set valid, false to set invalid. |

**70.7.36** `void XRDC2_SetPeriphAccessLockMode ( XRDC2_Type * base,  
xrdc2_periph_t periph, xrdc2_access_config_lock_t lockMode )`

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | XRDC2 peripheral base address.      |
| <i>periph</i>   | Which peripheral descriptor to set. |
| <i>lockMode</i> | The lock mode to set.               |

**70.7.37** `void XRDC2_SetPeriphDomainAccessPolicy ( XRDC2_Type * base,  
xrdc2_periph_t periph, uint8_t domainId, xrdc2_access_policy_t policy )`

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>base</i>     | XRDC2 peripheral base address.                     |
| <i>periph</i>   | The peripheral to operate.                         |
| <i>domainId</i> | The ID of the domain whose policy will be changed. |
| <i>policy</i>   | The access policy to set.                          |

**70.7.38** `void XRDC2_EnablePeriphExclAccessLock ( XRDC2_Type * base,  
xrdc2_periph_t periph, bool enable )`

Once disabled, it could not be enabled until reset.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | XRDC2 peripheral base address. |
|-------------|--------------------------------|

|               |                                   |
|---------------|-----------------------------------|
| <i>periph</i> | The peripheral to operate.        |
| <i>enable</i> | True to enable, false to disable. |

### 70.7.39 uint8\_t XRDC2\_GetPeriphExclAccessLockDomainOwner ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph* )

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | XRDC2 peripheral base address. |
| <i>periph</i> | The peripheral to operate.     |

Returns

The domain ID of the lock owner.

### 70.7.40 status\_t XRDC2\_TryLockPeriphExclAccess ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph* )

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | XRDC2 peripheral base address. |
| <i>periph</i> | The peripheral to operate.     |

Return values

|                        |                      |
|------------------------|----------------------|
| <i>kStatus_Fail</i>    | Failed to lock.      |
| <i>kStatus_Success</i> | Locked successfully. |

### 70.7.41 void XRDC2\_LockPeriphExclAccess ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph* )

Parameters

---

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | XRDC2 peripheral base address. |
| <i>periph</i> | The peripheral to operate.     |

Note

This function must be called when the lock is not disabled.

#### 70.7.42 void XRDC2\_UnlockPeriphExclAccess ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph* )

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | XRDC2 peripheral base address. |
| <i>periph</i> | The peripheral to operate.     |

Note

This function must be called by the lock owner.

#### 70.7.43 void XRDC2\_ForcePeriphExclAccessLockRelease ( XRDC2\_Type \* *base*, xrdc2\_periph\_t *periph* )

The master does not own the lock could call this function to force release the lock.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | XRDC2 peripheral base address. |
| <i>periph</i> | The peripheral to operate.     |

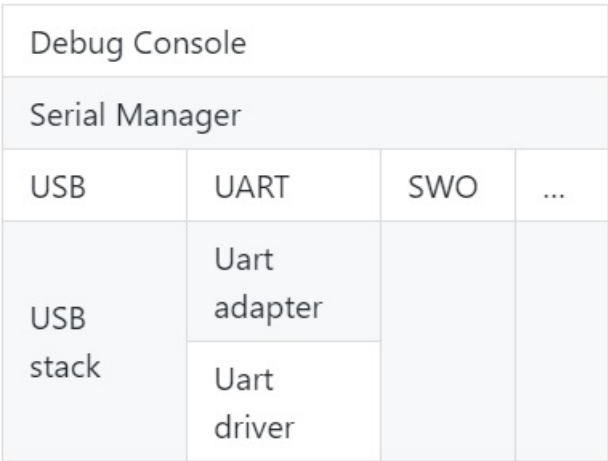
# Chapter 71

## Debug Console

### 71.1 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

### 71.2 Function groups

#### 71.2.1 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);
```

## 71.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype "`%[flags][width][.precision][length]specifier`", which is explained below

| flags   | Description                                                                                                                                                                                                                                                                                                                                                                             |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -       | Left-justified within the given field width. Right-justified is the default.                                                                                                                                                                                                                                                                                                            |
| +       | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.                                                                                                                                                                                                                                |
| (space) | If no sign is written, a blank space is inserted before the value.                                                                                                                                                                                                                                                                                                                      |
| #       | Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed. |
| 0       | Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).                                                                                                                                                                                                                                                                           |

| Width    | Description                                                                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (number) | A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger. |
| *        | The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                          |

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

| <b>length</b>  | <b>Description</b> |
|----------------|--------------------|
| Do not support |                    |

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



- 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 */

```

### 71.2.3 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_readc` will be used when `__REDLIB__` is defined; `_write` and `_read` will be used in other cases. The macro does not specifically refer to the peripheral "UART". It refers to the external peripheral similar to UART, like as USB CDC, UART, SWO, etc. So if the macro `SDK_DEBUGCONSOLE_UART` is not defined when tool-chain `printf` is calling, the semihosting will be used.

The following matrix shows the effects of `SDK_DEBUGCONSOLE` and `SDK_DEBUGCONSOLE_UART` on `PRINTF` and `printf`. The green mark is the default setting of the debug console.

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

\* the **low level peripheral** could be USB CDC, UART, or SWO, and so on.

## 71.3 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\r\nTime: %u ticks %2.5f milliseconds\r\n\r\nDONE\r\n", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

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

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

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

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file 'fsl\_sbrk.c' in path: ..\{package}\devices\{subset}\utilities\fsl-  
\_sbrk.c to your project.

## Modules

- [SWO](#)
- [Semihosting](#)

## 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.*

## Typedefs

- typedef void(\* [printfCb](#) )(char \*buf, int32\_t \*indicator, char val, int len)  
*A function pointer which is used when format printf log.*

## Functions

- int [StrFormatPrintf](#) (const char \*fmt, va\_list ap, char \*buf, [printfCb](#) cb)  
*This function outputs its parameters according to a formatted string.*
- int [StrFormatScanf](#) (const char \*line\_ptr, char \*format, va\_list args\_ptr)  
*Converts an input line of ASCII characters based upon a provided string format.*

## 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.*

## 71.4 Macro Definition Documentation

### 71.4.1 #define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U

Select toolchain printf and scanf.

### 71.4.2 #define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U

### 71.4.3 #define DEBUGCONSOLE\_DISABLE 2U

### 71.4.4 #define SDK\_DEBUGCONSOLE DEBUGCONSOLE\_REDIRECT\_TO\_SDK

The macro only support to be redefined in project setting.

### 71.4.5 #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.

## 71.5 Function Documentation

### 71.5.1 status\_t DbgConsole\_Init ( uint8\_t instance, uint32\_t baudRate, serial\_port\_type\_t device, uint32\_t clkSrcFreq )

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

## Parameters

|                   |                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>   | The instance of the module.If the device is kSerialPort_Uart, the instance is UART peripheral instance. The UART hardware peripheral type is determined by UART adapter. For example, if the instance is 1, if the lpuart_adapter.c is added to the current project, the UART peripheral is LPUART1. If the uart_adapter.c is added to the current project, the UART peripheral is UART1. |
| <i>baudRate</i>   | The desired baud rate in bits per second.                                                                                                                                                                                                                                                                                                                                                 |
| <i>device</i>     | Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"> <li>• kSerialPort_Uart,</li> <li>• kSerialPort_UsbCdc</li> </ul>                                                                                                                                                                                                             |
| <i>clkSrcFreq</i> | Frequency of peripheral source clock.                                                                                                                                                                                                                                                                                                                                                     |

## Returns

Indicates whether initialization was successful or not.

## Return values

|                        |                        |
|------------------------|------------------------|
| <i>kStatus_Success</i> | Execution successfully |
|------------------------|------------------------|

### 71.5.2 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.

### 71.5.3 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.

**71.5.4 status\_t DbgConsole\_ExitLowpower ( void )**

This function is used to restore from low power consumption.

Returns

Indicates whether de-initialization was successful or not.

**71.5.5 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.

**71.5.6 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.

**71.5.7 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.

### 71.5.8 int DbgConsole\_Scanf ( char \* *fmt\_s*, ... )

Call this function to read formatted data from the standard input stream.

## Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

## Parameters

|              |                        |
|--------------|------------------------|
| <i>fmt_s</i> | Format control string. |
|--------------|------------------------|

## Returns

Returns the number of fields successfully converted and assigned.

### 71.5.9 int DbgConsole\_Getchar ( void )

Call this function to read a character from standard input.

## Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

## Returns

Returns the character read.

### 71.5.10 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.

### 71.5.11 int DbgConsole\_BlockingVprintf ( const char \* *fmt\_s*, va\_list *formatStringArg* )

Call this function to write a formatted output to the standard output stream with the blocking mode. The function will send data with blocking mode no matter the DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set or not. The function could be used in system ISR mode with DEBUG\_CONSOLE\_TRANSFER\_NON\_BLOCKING set.

Parameters

|                        |                        |
|------------------------|------------------------|
| <i>fmt_s</i>           | Format control string. |
| <i>formatStringArg</i> | Format arguments.      |

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 71.5.12 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.

**71.5.13 int StrFormatPrintf ( const char \* *fmt*, va\_list *ap*, char \* *buf*, printfCb *cb* )**

Note

I/O is performed by calling given function pointer using following (\*func\_ptr)(c);

Parameters

|    |            |                                |
|----|------------|--------------------------------|
| in | <i>fmt</i> | Format string for printf.      |
| in | <i>ap</i>  | Arguments to printf.           |
| in | <i>buf</i> | pointer to the buffer          |
|    | <i>cb</i>  | print callbck function pointer |

Returns

Number of characters to be print

**71.5.14 int StrFormatScanf ( const char \* *line\_ptr*, char \* *format*, va\_list *args\_ptr* )**

Parameters

|    |                 |                                           |
|----|-----------------|-------------------------------------------|
| in | <i>line_ptr</i> | The input line of ASCII data.             |
| in | <i>format</i>   | Format first points to the format string. |
| in | <i>args_ptr</i> | The list of parameters.                   |

Returns

Number of input items converted and assigned.

Return values

|               |                                   |
|---------------|-----------------------------------|
| <i>IO_EOF</i> | When line_ptr is empty string "". |
|---------------|-----------------------------------|

## 71.6 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

### 71.6.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is `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.

### 71.6.2 Guide Semihosting for Keil $\mu$ Vision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

### 71.6.3 Guide Semihosting for MCUXpresso IDE

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK\_DEBUGCONSOLE=0, if set SDK\_DEBUGCONSOLE=1, the log will be redirect to the UART.

#### Step 2: Building the project

1. Compile and link the project.

#### Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

### 71.6.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

## Step 2: Building the project

1. Change "CMakeLists.txt":

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

**Replace paragraph**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-common")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffunction-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fdata-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffreestanding")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-builtin")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mthumb")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mapcs")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --gc-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -static")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -z")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} muldefs")

**To**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --specs=rdimon.specs ")

**Remove**

target\_link\_libraries(semihosting\_ARMGCC.elf debug nosys)

2. Run "build\_debug.bat" to build project

### Step 3: Starting semihosting

1. Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

2. After the setting, press "enter". The PuTTY window now shows the printf() output.

## 71.7 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.

### 71.7.1 Guide SWO for SDK

**NOTE:** After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

#### Step 1: Setting up the environment

1. Define 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

##### 71.7.1.1 Guide SWO for IAR

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

#### Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK, and click the Activate button to enable trace data collection.
6. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, The SDK\_DEBUGCONSOLE\_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then debug function ok. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then debug function ok.



**NOTE:** Case a or c only apply at example which enable swo function,the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. Start the project by choosing Project>Download and Debug.

### **Step 2: Building the project**

### **Step 3: Starting swo**

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

## **71.7.2 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 O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

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

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

### **Step 4: Run the project**

1. Choose "Debug" on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

### 71.7.3 Guide SWO for MCUXpresso IDE

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### 71.7.4 Guide SWO for ARMGCC

**NOTE:** ARMGCC has no library support SWO.

## Chapter 72

# Notification Framework

### 72.1 Overview

This section describes the programming interface of the Notifier driver.

### 72.2 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.*

## 72.3 Data Structure Documentation

### 72.3.1 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`

### 72.3.2 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`

### 72.3.3 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](#)

## 72.4 Typedef Documentation

### 72.4.1 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.

### 72.4.2 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`.

### 72.4.3 `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`.

## 72.5 Enumeration Type Documentation

### 72.5.1 `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.



### 72.5.2 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.

### 72.5.3 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.

### 72.5.4 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.

## 72.6 Function Documentation

**72.6.1** `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.

### 72.6.2 **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.

### 72.6.3 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 73

## Shell

### 73.1 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.

### 73.2 Function groups

#### 73.2.1 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>");
```

#### 73.2.2 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.                     |

#### 73.2.3 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, paramCount)  
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) }  
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.*

## 73.3 Data Structure Documentation

### 73.3.1 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**

## 73.4 Macro Definition Documentation

**73.4.1 #define SHELL\_NON\_BLOCKING\_MODE SERIAL\_MANAGER\_NON\_BLOCKING\_MODE**

**73.4.2 #define SHELL\_AUTO\_COMPLETE (1U)**

**73.4.3 #define SHELL\_BUFFER\_SIZE (64U)**

**73.4.4 #define SHELL\_MAX\_ARGS (8U)**

**73.4.5 #define SHELL\_HISTORY\_COUNT (3U)**

**73.4.6 #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

**73.4.7 #define SHELL\_USE\_COMMON\_TASK (0U)**

**73.4.8 #define SHELL\_TASK\_PRIORITY (2U)**

**73.4.9 #define SHELL\_TASK\_STACK\_SIZE (1000U)**



### 73.4.10 **#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. |
|-------------|--------------------------------------|

### 73.4.11 **#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.                                                   |

### 73.4.12 #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". |
|----------------|------------------------------------------------------------------------------------------------------------------------------|

## 73.5 Typedef Documentation

### 73.5.1 typedef shell\_status\_t(\* cmd\_function\_t)(shell\_handle\_t shellHandle, int32\_t argc, char \*\*argv)

## 73.6 Enumeration Type Documentation

### 73.6.1 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.

## 73.7 Function Documentation

### 73.7.1 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 <a href="#">SHELL_HANDLE_SIZE</a> allocated by the caller. The handle should be 4 byte aligned, because unaligned access doesn't be supported on some devices. You can define the handle in the following two ways: <a href="#">SHELL_HANDLE_DEFINE(shellHandle)</a> ; 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.                     |

### 73.7.2 `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.                 |

### 73.7.3 **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. |
|------------------------------|--------------------------------------|

### 73.7.4 **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.      |

### 73.7.5 **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.

### 73.7.6 **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.      |

### 73.7.7 **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.

### 73.7.8 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.

### 73.7.9 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.

### 73.7.10 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. |
|--------------------|----------------------------------|

**73.7.11 static bool SHELL\_checkRunningInIsr ( void ) [inline], [static]**

This function is used to check if code running in ISR.

## Return values

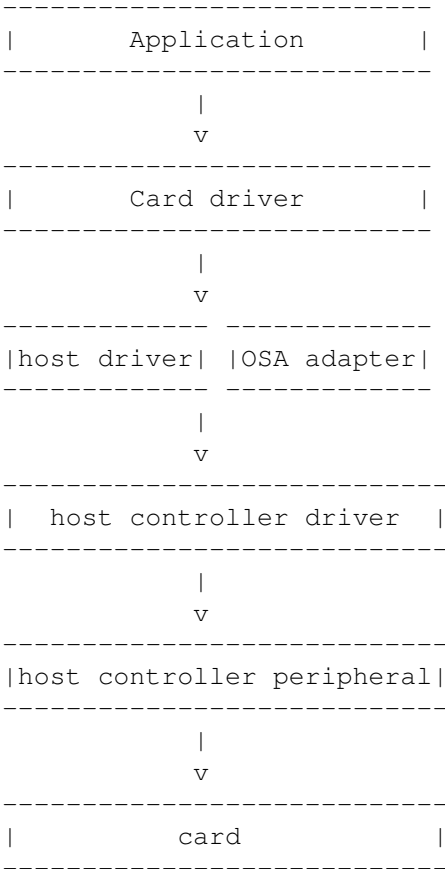
|             |                        |
|-------------|------------------------|
| <i>TRUE</i> | if code runing in ISR. |
|-------------|------------------------|

# Chapter 74

## Cards: Secure Digital Card/Embedded MultiMedia Card/SD-IO Card

### 74.1 Overview

The MCUXpresso SDK provides drivers to access the Secure Digital Card(up to v3.0), Embedded MultiMedia 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](#)



## 74.2 SDIO Card Driver

### 74.2.1 Overview

The SDIO card driver provide card initialization/IO direct and extend command interface.

### 74.2.2 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, 0U)`) /\*2.4.0\*/  
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.*

## 74.2.3 Data Structure Documentation

### 74.2.3.1 struct \_sdio\_card

sdio card descriptor

Define the card structure including the necessary fields to identify and describe the card.

## Data Fields

- `sdrmchost_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*
- `sdmhc_osa_mutex_t lock`  
*card access lock*

## Field Documentation

### (1) `bool sdio_card_t::noInternalAlign`

If disable, sdmhc will not make sure the data buffer address is word align, otherwise all the transfer are align to low level driver

## 74.2.4 Macro Definition Documentation

**74.2.4.1** `#define FSL_SDIO_DRIVER_VERSION (MAKE_VERSION(2U, 4U, 0U)) /*2.4.0*/`

## 74.2.5 Enumeration Type Documentation

### 74.2.5.1 `enum sdio_io_direction_t`

Enumerator

*kSDIO\_IORead* io read  
*kSDIO\_IOWrite* io write

## 74.2.6 Function Documentation

### 74.2.6.1 `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 below 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>                                   |  |

#### 74.2.6.2 void SDIO\_Deinit ( sdio\_card\_t \* *card* )

Please note it is a thread safe function.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.2.6.3 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_Set-CardBlockSizeFailed</i> | Set card block size failed. |
| <i>kStatus_Success</i>                       | Operate successfully.       |

#### 74.2.6.4 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. |
|-------------|------------------|

#### 74.2.6.5 status\_t SDIO\_HostInit ( sdio\_card\_t \* card )

This function deinitializes the specific host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.2.6.6 void SDIO\_HostDeinit ( sdio\_card\_t \* card )

This function deinitializes the host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.2.6.7 void SDIO\_HostDoReset ( sdio\_card\_t \* card )

This function reset the specific host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.2.6.8 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. |

**74.2.6.9 status\_t SDIO\_CardIsActive ( sdio\_card\_t \* *card* )**

## Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

## Return values

|                                      |  |
|--------------------------------------|--|
| <i>kStatus_SDMMC_-TransferFailed</i> |  |
| <i>kStatus_Success</i>               |  |

**74.2.6.10 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>               |  |

**74.2.6.11 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_Set-CardBlockSizeFailed</i> |  |
| <i>kStatus_SDMMC_SDIO-InvalidArgument</i>    |  |
| <i>kStatus_Success</i>                       |  |

#### 74.2.6.12 **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>              |  |

#### 74.2.6.13 **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>              |  |

**74.2.6.14** `status_t SDIO_SwitchToHighSpeed ( sdio_card_t * card )`

## Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

## Return values

|                                                     |  |
|-----------------------------------------------------|--|
| <i>kStatus_SDMMC_-<br/>TransferFailed</i>           |  |
| <i>kStatus_SDMMC_SDIO-<br/>_SwitchHighSpeedFail</i> |  |
| <i>kStatus_Success</i>                              |  |

**74.2.6.15** `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-<br/>_ReadCISFail</i> |  |
| <i>kStatus_SDMMC_-<br/>TransferFailed</i>   |  |
| <i>kStatus_Success</i>                      |  |

**74.2.6.16** `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. |

#### 74.2.6.17 bool SDIO\_IsCardPresent ( sdio\_card\_t \* *card* )

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | card descriptor. |
|-------------|------------------|

#### 74.2.6.18 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 numner                                     |
| <i>regAddr</i> | register address                              |
| <i>data</i>    | the data pinter to write                      |
| <i>raw</i>     | flag, indicate read after write or write only |

Return values

|                                      |  |
|--------------------------------------|--|
| <i>kStatus_SDMMC_-TransferFailed</i> |  |
| <i>kStatus_Success</i>               |  |

#### 74.2.6.19 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>               |  |

**74.2.6.20** `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 sdio_io_direction_t.                                                                                                |
| <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, dataOut should not be NULL. |

Return values

|                                      |  |
|--------------------------------------|--|
| <i>kStatus_SDMMC_-TransferFailed</i> |  |
| <i>kStatus_Success</i>               |  |

**74.2.6.21** `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>                     |  |

**74.2.6.22** `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>                     |  |

**74.2.6.23** `status_t SDIO_EnableInterrupt ( 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>               |  |

#### 74.2.6.24 **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>               |  |

#### 74.2.6.25 **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>               |  |



**74.2.6.26** `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>               |  |

#### 74.2.6.27 **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>               |  |

#### 74.2.6.28 **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> |  |
|------------------------|--|

#### 74.2.6.29 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>              |  |

#### 74.2.6.30 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> | response pointer, if application want read response back, please set it to a NON-NULL pointer. |

#### 74.2.6.31 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.     |

#### 74.2.6.32 status\_t SDIO\_HandlePendingIOInterrupt ( sdio\_card\_t \* *card* )

This function is used to handle the pending io interrupt. To register a IO IRQ handler,

```
* SDIO_EnableIOInterrupt(card, 0, true);
* SDIO_SetIOIRQHandler(card, 0, func0_handler);
*
```

call it in interrupt callback

```
* SDIO_HandlePendingIOInterrupt(card);
*
```

To release 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_-<br/>TransferFailed</i> |  |
| <i>kStatus_Success</i>                    |  |

## 74.3 SD Card Driver

### 74.3.1 Overview

The SDCARD driver provide card initialization/read/write/erase interface.

### 74.3.2 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
{
 sdmmc_host_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_sdmmcHostDmaBuffer;
s_host.dmaDesBufferWordsNum = xxx;
/* */
((sd_card_t *)card)->host = &s_host;
((sd_card_t *)card)->host->hostController.base = BOARD_SDMMC_SD_HOST_BASEADDR;
((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, 0U)) /\*2.4.0\*/  
*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 block-Count)



- *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*
- `status_t SD_SetMaxCurrent` (`sd_card_t *card`, `sd_max_current_t maxCurrent`)
- *select max current select max operation current*
- `status_t SD_PollingCardStatusBusy` (`sd_card_t *card`, `uint32_t timeoutMs`)
- *Polling card idle status.*

### 74.3.3 Data Structure Documentation

#### 74.3.3.1 struct sd\_card\_t

Define the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- `sdmchost_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 united 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*

## 74.3.4 Macro Definition Documentation

74.3.4.1 #define FSL\_SD\_DRIVER\_VERSION (MAKE\_VERSION(2U, 4U, 0U)) /\*2.4.0\*/

## 74.3.5 Enumeration Type Documentation

### 74.3.5.1 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

## 74.3.6 Function Documentation

### 74.3.6.1 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_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_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_Set-DataBusWidthFailed</i>              | Set bus width failed.            |
| <i>kStatus_SDMMC_Switch-BusTimingFailed</i>              | Switch high speed failed.        |
| <i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>             | Set card block size failed.      |

|                        |                       |
|------------------------|-----------------------|
| <i>kStatus_Success</i> | Operate successfully. |
|------------------------|-----------------------|

#### 74.3.6.2 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. |
|-------------|------------------|

#### 74.3.6.3 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_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_Set-DataBusWidthFailed</i>     | Set bus width failed.            |
| <i>kStatus_SDMMC_Switch-BusTimingFailed</i>     | Switch high speed failed.        |
| <i>kStatus_SDMMC_Set-CardBlockSizeFailed</i>    | Set card block size failed.      |
| <i>kStatus_Success</i>                          | Operate successfully.            |

#### 74.3.6.4 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. |
|-------------|------------------|

#### 74.3.6.5 status\_t SD\_HostInit ( sd\_card\_t \* *card* )

This function deinitializes the specific host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.3.6.6 void SD\_HostDeinit ( sd\_card\_t \* *card* )

This function deinitializes the host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.3.6.7 void SD\_HostDoReset ( sd\_card\_t \* *card* )

This function reset the specific host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.3.6.8 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. |

#### 74.3.6.9 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. |

#### 74.3.6.10 bool SD\_IsCardPresent ( sd\_card\_t \* *card* )

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | card descriptor. |
|-------------|------------------|

#### 74.3.6.11 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. |

**74.3.6.12 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. |

**74.3.6.13 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_Send-ApplicationCommand-Failed</i> | send application command failed. |

|                        |                       |
|------------------------|-----------------------|
| <i>kStatus_Success</i> | Operate successfully. |
|------------------------|-----------------------|

#### 74.3.6.14 **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.     |

#### 74.3.6.15 **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.     |

#### 74.3.6.16 `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_Wait-WriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_SDMMC_-TransferFailed</i>          | Transfer failed.      |
| <i>kStatus_SDMMC_Wait-WriteCompleteFailed</i> | Send status failed.   |
| <i>kStatus_Success</i>                        | Operate successfully. |

#### 74.3.6.17 **status\_t SD\_SetDriverStrength ( sd\_card\_t \* *card*, sd\_driver\_strength\_t *driverStrength* )**

Parameters

|                       |                  |
|-----------------------|------------------|
| <i>card</i>           | Card descriptor. |
| <i>driverStrength</i> | Driver strength  |

#### 74.3.6.18 **status\_t SD\_SetMaxCurrent ( sd\_card\_t \* *card*, sd\_max\_current\_t *maxCurrent* )**

Parameters

|                   |                  |
|-------------------|------------------|
| <i>card</i>       | Card descriptor. |
| <i>maxCurrent</i> | Max current      |

#### 74.3.6.19 **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. |

## 74.4 MMC Card Driver

### 74.4.1 Overview

The MMCCARD driver provide card initialization/read/write/erase interface.

### 74.4.2 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.*
- enum [mmc\\_sleep\\_awake\\_t](#) {  
[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](#) (SDMMC\_HOST\_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.*

### 74.4.3 Data Structure Documentation

#### 74.4.3.1 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*

### 74.4.3.2 struct mmc\_card\_t

Defines the card structure including the necessary fields to identify and describe the card.

#### Data Fields

- `sdrmchost_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::noInterAlAlign`

If disabled, sdmmc will not make sure the data buffer address is word align, otherwise all the transfer are aligned to low level driver.

## 74.4.4 Macro Definition Documentation

**74.4.4.1** `#define FSL_MMC_DRIVER_VERSION (MAKE_VERSION(2U, 5U, 0U)) /*2.5.0*/`

## 74.4.5 Enumeration Type Documentation

### 74.4.5.1 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

### 74.4.5.2 `enum mmc_sleep_aware_t`

Enumerator

*kMMC\_Sleep* MMC card sleep.  
*kMMC\_Awake* MMC card awake.

## 74.4.6 Function Documentation



**74.4.6.1** `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_Host-NotReady</i>                       | Host is not ready.                  |
| <i>kStatus_SDMMC_Go-IdleFailed</i>                       | Going idle failed.                  |
| <i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i> | Sending operation condition failed. |
| <i>kStatus_SDMMC_All-SendCidFailed</i>                   | Sending CID failed.                 |
| <i>kStatus_SDMMC_Set-RelativeAddressFailed</i>           | Setging relative address failed.    |
| <i>kStatus_SDMMC_Send-CsdFailed</i>                      | Sending CSD failed.                 |
| <i>kStatus_SDMMC_Card-NotSupport</i>                     | Card not support.                   |
| <i>kStatus_SDMMC_Select-CardFailed</i>                   | Sending SELECT_CARD command failed. |
| <i>kStatus_SDMMC_Send-ExtendedCsdFailed</i>              | Sending EXT_CSD failed.             |
| <i>kStatus_SDMMC_Set-DataBusWidthFailed</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.             |

#### 74.4.6.2 void MMC\_Deinit ( mmc\_card\_t \* *card* )

Note

It is a thread safe function.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.4.6.3 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_Go-IdleFailed</i>                       | Going idle failed.                  |
| <i>kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed</i> | Sending operation condition failed. |
| <i>kStatus_SDMMC_All-SendCidFailed</i>                   | Sending CID failed.                 |
| <i>kStatus_SDMMC_Set-RelativeAddressFailed</i>           | Setting relative address failed.    |
| <i>kStatus_SDMMC_Send-CsdFailed</i>                      | Sending CSD failed.                 |
| <i>kStatus_SDMMC_Card-NotSupport</i>                     | Card not support.                   |
| <i>kStatus_SDMMC_Select-CardFailed</i>                   | Sending SELECT_CARD command failed. |
| <i>kStatus_SDMMC_Send-ExtendedCsdFailed</i>              | Sending EXT_CSD failed.             |
| <i>kStatus_SDMMC_Set-DataBusWidthFailed</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.                |

#### 74.4.6.4 void MMC\_CardDeinit ( mmc\_card\_t \* card )

Note

It is a thread safe function.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.4.6.5 **status\_t MMC\_HostInit ( mmc\_card\_t \* *card* )**

This function deinitializes the specific host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.4.6.6 **void MMC\_HostDeinit ( mmc\_card\_t \* *card* )**

This function deinitializes the host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.4.6.7 **void MMC\_HostDoReset ( mmc\_card\_t \* *card* )**

This function resets the specific host.

Parameters

|             |                  |
|-------------|------------------|
| <i>card</i> | Card descriptor. |
|-------------|------------------|

#### 74.4.6.8 **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. |
|-------------|------------------|

#### 74.4.6.9 **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. |

**74.4.6.10 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. |

**74.4.6.11 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.          |

#### 74.4.6.12 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.      |

#### 74.4.6.13 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. |

#### 74.4.6.14 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.        |

**74.4.6.15** `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.        |

**74.4.6.16** `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_GoIdleFailed</i>   | Resetting card failed. |
| <i>kStatus_Success</i>              | Operation succeeded.   |

#### 74.4.6.17 status\_t MMC\_SetBootConfigWP ( mmc\_card\_t \* *card*, uint8\_t *wp* )

## Parameters

|             |                      |
|-------------|----------------------|
| <i>card</i> | Card descriptor.     |
| <i>wp</i>   | Write protect value. |

#### 74.4.6.18 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. |

#### 74.4.6.19 status\_t MMC\_StopBoot ( mmc\_card\_t \* *card*, uint32\_t *bootMode* )

## Parameters

|                 |                  |
|-----------------|------------------|
| <i>card</i>     | Card descriptor. |
| <i>bootMode</i> | Boot mode.       |

#### 74.4.6.20 **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. |

#### 74.4.6.21 **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. |

#### 74.4.6.22 **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. |
|-------------|------------------|

#### 74.4.6.23 **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 <a href="#">mmc_sleep_awake_t</a> . |

Return values

|                                             |                                                                      |
|---------------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SDMMC_Not-SupportYet</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_Select-CardFailed</i>      | Indicates select card command failed.                                |
| <i>kStatus_Success</i>                      | Indicates the card state switched successfully.                      |

#### 74.4.6.24 **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-<br/>StatusIdle</i> | Card is idle.                        |
| <i>kStatus_SDMMC_Card-<br/>StatusBusy</i> | Card is busy.                        |
| <i>kStatus_SDMMC_-<br/>TransferFailed</i> | Command transfer failed.             |
| <i>kStatus_SDMMC_Switch-<br/>Failed</i>   | Status command reports switch error. |

## 74.5 SDMMC HOST Driver

### 74.5.1 Overview

The host adapter driver provide adapter for blocking/non\_blocking mode.

#### Modules

- [USDHC HOST adapter Driver](#)

## 74.6 SDMMC OSA

### 74.6.1 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 timeout-Milliseconds, 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\\_OSADelay](#) (uint32\_t milliseconds)  
*sdmmc delay.*
- uint32\_t [SDMMC\\_OSADelayUs](#) (uint32\_t microseconds)  
*sdmmc delay us.*

## 74.6.2 Data Structure Documentation

### 74.6.2.1 struct sdmmc\_osa\_event\_t

### 74.6.2.2 struct sdmmc\_osa\_mutex\_t

## 74.6.3 Function Documentation

### 74.6.3.1 status\_t SDMMC\_OSAEventCreate ( void \* *eventHandle* )

Parameters

|                    |               |
|--------------------|---------------|
| <i>eventHandle</i> | event handle. |
|--------------------|---------------|

Return values

|                     |                     |
|---------------------|---------------------|
| <i>kStatus_Fail</i> | or kStatus_Success. |
|---------------------|---------------------|

### 74.6.3.2 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 kStatus_Success. |
|---------------------|---------------------|

#### 74.6.3.3 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 kStatus_Success. |
|---------------------|---------------------|

#### 74.6.3.4 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 kStatus_Success. |
|---------------------|---------------------|

#### 74.6.3.5 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 kStatus_Success. |
|---------------------|---------------------|

#### 74.6.3.6 status\_t SDMMC\_OSAEventDestroy ( void \* *eventHandle* )

Parameters

|                    |                   |
|--------------------|-------------------|
| <i>eventHandle</i> | The event handle. |
|--------------------|-------------------|

#### 74.6.3.7 status\_t SDMMC\_OSAMutexCreate ( void \* *mutexHandle* )

Parameters

|                    |               |
|--------------------|---------------|
| <i>mutexHandle</i> | mutex handle. |
|--------------------|---------------|

Return values

|                     |                     |
|---------------------|---------------------|
| <i>kStatus_Fail</i> | or kStatus_Success. |
|---------------------|---------------------|

#### 74.6.3.8 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 osaWaitForever_c will wait indefinitely, pass 0 will return KOSA_StatusTimeout immediately. |

Return values

|                     |                     |
|---------------------|---------------------|
| <i>kStatus_Fail</i> | or kStatus_Success. |
|---------------------|---------------------|

#### 74.6.3.9 status\_t SDMMC\_OSAMutexUnlock ( void \* *mutexHandle* )

## Parameters

|                    |               |
|--------------------|---------------|
| <i>mutexHandle</i> | mutex handle. |
|--------------------|---------------|

## Return values

|                     |                     |
|---------------------|---------------------|
| <i>kStatus_Fail</i> | or kStatus_Success. |
|---------------------|---------------------|

**74.6.3.10 status\_t SDMMC\_OSAMutexDestroy ( void \* *mutexHandle* )**

## Parameters

|                    |                   |
|--------------------|-------------------|
| <i>mutexHandle</i> | The mutex handle. |
|--------------------|-------------------|

**74.6.3.11 void SDMMC\_OSADelay ( uint32\_t *milliseconds* )**

## Parameters

|                     |               |
|---------------------|---------------|
| <i>milliseconds</i> | time to delay |
|---------------------|---------------|

**74.6.3.12 uint32\_t SDMMC\_OSADelayUs ( uint32\_t *microseconds* )**

## Parameters

|                     |               |
|---------------------|---------------|
| <i>microseconds</i> | time to delay |
|---------------------|---------------|

## Returns

actual delayed microseconds

## 74.6.4 USDHC HOST adapter Driver

### 74.6.4.1 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  $2 * \text{cache line size}$ . 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, 1U)) /*2.6.1*/`  
*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 `SDMMCCHOST_INSTANCE_SUPPORT_8_BIT_WIDTH`(host) FSL\_FEATURE\_USDHC\_INSTANCE\_SUPPORT\_8\_BIT\_WIDTHn(host->hostController.base)  
*sdmmc host instance capability*
- #define `SDMMCCHOST_DATA3_DETECT_CARD_DELAY` (10U)  
*sdmmc host delay for DAT3 detect card*
- #define `SDMMCCHOST_DMA_DESCRIPTOR_BUFFER_ALIGN_SIZE` (4U)  
*SDMMC host dma descriptor buffer address align size.*
- #define `SDMMCCHOST_STANDARD_TUNING_START` (10U)  
*tuning configuration*
- #define `SDMMCCHOST_TUINIG_STEP` (2U)  
*standard tuning stBep*

## Typedefs

- typedef `usdhc_transfer_t` `sdmmcchost_transfer_t`  
*sdmmc host transfer function*

## Enumerations

- enum {  
`kSDMMCCHOST_SupportHighSpeed` = 1U << 0U,  
`kSDMMCCHOST_SupportSuspendResume` = 1U << 1U,  
`kSDMMCCHOST_SupportVoltage3v3` = 1U << 2U,  
`kSDMMCCHOST_SupportVoltage3v0` = 1U << 3U,  
`kSDMMCCHOST_SupportVoltage1v8` = 1U << 4U,  
`kSDMMCCHOST_SupportVoltage1v2` = 1U << 5U,  
`kSDMMCCHOST_Support4BitDataWidth` = 1U << 6U,  
`kSDMMCCHOST_Support8BitDataWidth` = 1U << 7U,  
`kSDMMCCHOST_SupportDDRMMode` = 1U << 8U,  
`kSDMMCCHOST_SupportDetectCardByData3` = 1U << 9U,  
`kSDMMCCHOST_SupportDetectCardByCD` = 1U << 10U,  
`kSDMMCCHOST_SupportAutoCmd12` = 1U << 11U,  
`kSDMMCCHOST_SupportSDR104` = 1U << 12U,  
`kSDMMCCHOST_SupportSDR50` = 1U << 13U,  
`kSDMMCCHOST_SupportHS200` = 1U << 14U,  
`kSDMMCCHOST_SupportHS400` = 1U << 15U }  
*sdmmc host capability*
- enum {  
`kSDMMCCHOST_EndianModeBig` = 0U,  
`kSDMMCCHOST_EndianModeHalfWordBig` = 1U,  
`kSDMMCCHOST_EndianModeLittle` = 2U }  
*host Endian mode corresponding to driver define*
- enum {  
`kSDMMCCHOST_StandardTuning` = 0U,  
`kSDMMCCHOST_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](#) (sdmmchost\_t \*host, uint32\_t dataBusWidth)  
     set data bus width.
- static void [SDMMCHOST\\_SendCardActive](#) (sdmmchost\_t \*host)  
     Send initialization active 80 clocks to card.
- static uint32\_t [SDMMCHOST\\_SetCardClock](#) (sdmmchost\_t \*host, uint32\_t targetClock)  
     Set card bus clock.
- static bool [SDMMCHOST\\_IsCardBusy](#) (sdmmchost\_t \*host)  
     check card status by DATA0.
- static uint32\_t [SDMMCHOST\\_GetSignalLineStatus](#) (sdmmchost\_t \*host, uint32\_t signalLine)  
     Get signal line status.
- static void [SDMMCHOST\\_EnableCardInt](#) (sdmmchost\_t \*host, bool enable)  
     enable card interrupt.
- static void [SDMMCHOST\\_EnableDDRMode](#) (sdmmchost\_t \*host, bool enable, uint32\_t nibble-Pos)  
     enable DDR mode.
- static void [SDMMCHOST\\_EnableHS400Mode](#) (sdmmchost\_t \*host, bool enable)  
     enable HS400 mode.
- static void [SDMMCHOST\\_EnableStrobeDll](#) (sdmmchost\_t \*host, bool enable)  
     enable STROBE DLL.
- status\_t [SDMMCHOST\\_StartBoot](#) (sdmmchost\_t \*host, sdmmchost\_boot\_config\_t \*hostConfig, sdmmchost\_cmd\_t \*cmd, uint8\_t \*buffer)  
     start read boot data.
- status\_t [SDMMCHOST\\_ReadBootData](#) (sdmmchost\_t \*host, sdmmchost\_boot\_config\_t \*hostConfig, uint8\_t \*buffer)  
     read boot data.
- static void [SDMMCHOST\\_EnableBoot](#) (sdmmchost\_t \*host, bool enable)  
     enable boot mode.
- status\_t [SDMMCHOST\\_CardIntInit](#) (sdmmchost\_t \*host, void \*sdioInt)  
     card interrupt function.
- static void [SDMMCHOST\\_ForceClockOn](#) (sdmmchost\_t \*host, bool enable)  
     force card clock on.
- void [SDMMCHOST\\_SwitchToVoltage](#) (sdmmchost\_t \*host, uint32\_t voltage)  
     switch to voltage.
- status\_t [SDMMCHOST\\_CardDetectInit](#) (sdmmchost\_t \*host, void \*cd)  
     card detect init function.
- status\_t [SDMMCHOST\\_PollingCardDetectStatus](#) (sdmmchost\_t \*host, uint32\_t waitCardStatus, uint32\_t timeout)  
     Detect card insert, only need for SD cases.
- uint32\_t [SDMMCHOST\\_CardDetectStatus](#) (sdmmchost\_t \*host)  
     card detect status.
- status\_t [SDMMCHOST\\_Init](#) (sdmmchost\_t \*host)  
     Init host controller.
- void [SDMMCHOST\\_Deinit](#) (sdmmchost\_t \*host)

- *Deinit host controller.*
- void [SDMMCHOST\\_SetCardPower](#) ([sdmmcchost\\_t](#) \*host, bool enable)  
*host power off card function.*
- [status\\_t SDMMCHOST\\_TransferFunction](#) ([sdmmcchost\\_t](#) \*host, [sdmmcchost\\_transfer\\_t](#) \*content)  
*host transfer function.*
- [status\\_t SDMMCHOST\\_ExecuteTuning](#) ([sdmmcchost\\_t](#) \*host, [uint32\\_t](#) tuningCmd, [uint32\\_t](#) \*rev-  
Buf, [uint32\\_t](#) blockSize)  
*sdmmc host excute tuning.*
- void [SDMMCHOST\\_Reset](#) ([sdmmcchost\\_t](#) \*host)  
*host reset function.*
- void [SDMMCHOST\\_ConvertDataToLittleEndian](#) ([sdmmcchost\\_t](#) \*host, [uint32\\_t](#) \*data, [uint32\\_t](#) wordSize, [uint32\\_t](#) format)  
*sdmmc host convert data sequence to little endian sequence*

## 74.6.4.2 Data Structure Documentation

### 74.6.4.2.1 struct [sdmmcchost\\_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*
- [uint8\\_t](#) [enableCacheControl](#)  
*Cache maintain flag in host driver.*
- [sdmmc\\_osa\\_mutex\\_t](#) lock  
*host access lock*

#### Field Documentation

**(1) uint8\_t sdmmchost\_t::enableCacheControl**

Host driver only maintain cache when FSL\_SDK\_ENABLE\_DRIVER\_CACHE\_CONTROL is not defined and enableCacheControl = kSDMMCHOST\_CacheControlRWBuffer. While FSL\_SDK\_ENABLE\_DRIVER\_CACHE\_CONTROL is defined, host driver will not maintain cache and peripheral driver will do it.

**74.6.4.3 Macro Definition Documentation**

**74.6.4.3.1 #define FSL\_SDMMC\_HOST\_ADAPTER\_VERSION (MAKE\_VERSION(2U, 6U, 1U))**  
/\*2.6.1\*/

**74.6.4.3.2 #define SDMMCHOST\_STANDARD\_TUNING\_START (10U)**

standard tuning start point

**74.6.4.4 Enumeration Type Documentation****74.6.4.4.1 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.

**74.6.4.4.2 anonymous enum**

Enumerator

*kSDMMCHOST\_EndianModeBig* Big endian mode.  
*kSDMMCHOST\_EndianModeHalfWordBig* Half word big endian mode.



***kSDMMCHOST\_EndianModeLittle*** Little endian mode.

#### 74.6.4.4.3 anonymous enum

Enumerator

***kSDMMCHOST\_StandardTuning*** standard tuning type

***kSDMMCHOST\_ManualTuning*** manual tuning type

#### 74.6.4.4.4 anonymous enum

Enumerator

***kSDMMCHOST\_NoCacheControl*** sdmmc host cache control disabled

***kSDMMCHOST\_CacheControlRWBuffer*** sdmmc host cache control read/write buffer

### 74.6.4.5 Function Documentation

**74.6.4.5.1 void SDMMCHOST\_SetCardBusWidth ( sdmmchost\_t \* *host*, uint32\_t *dataBusWidth* )**

Parameters

|                     |                |
|---------------------|----------------|
| <i>host</i>         | host handler   |
| <i>dataBusWidth</i> | data bus width |

**74.6.4.5.2 static void SDMMCHOST\_SendCardActive ( sdmmchost\_t \* *host* ) [inline], [static]**

Parameters

|             |              |
|-------------|--------------|
| <i>host</i> | host handler |
|-------------|--------------|

**74.6.4.5.3 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. |
|---------------|-------------------------------|

**74.6.4.5.4 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. |
|-------------|-------------------------|

**74.6.4.5.5 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 |

**74.6.4.5.6 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. |

**74.6.4.5.7 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'. |

**74.6.4.5.8 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. |

**74.6.4.5.9 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. |

**74.6.4.5.10 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     |

**74.6.4.5.11 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     |

**74.6.4.5.12 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 |

**74.6.4.5.13 status\_t SDMMCHOST\_CardIntInit ( sdmmchost\_t \* *host*, void \* *sdioint* )**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>host</i>    | host handler                 |
| <i>sdioint</i> | card interrupt configuration |

**74.6.4.5.14 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. |

**74.6.4.5.15 void SDMMCHOST\_SwitchToVoltage ( sdmmchost\_t \* *host*, uint32\_t *voltage* )**

## Parameters

---

|                |                          |
|----------------|--------------------------|
| <i>host</i>    | host handler             |
| <i>voltage</i> | switch to voltage level. |

#### 74.6.4.5.16 **status\_t SDMMCHOST\_CardDetectInit ( sdmmchost\_t \* *host*, void \* *cd* )**

Parameters

|             |                           |
|-------------|---------------------------|
| <i>host</i> | host handler              |
| <i>cd</i>   | card detect configuration |

#### 74.6.4.5.17 **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 |

#### 74.6.4.5.18 **uint32\_t SDMMCHOST\_CardDetectStatus ( sdmmchost\_t \* *host* )**

Parameters

|             |              |
|-------------|--------------|
| <i>host</i> | host handler |
|-------------|--------------|

Return values

|                                  |  |
|----------------------------------|--|
| <i>kSD_Inserted, kSD_Removed</i> |  |
|----------------------------------|--|

**74.6.4.5.19 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        |

**74.6.4.5.20 void SDMMCHOST\_Deinit ( sdmmchost\_t \* *host* )**

Please note it is a thread safe function.

Parameters

|             |              |
|-------------|--------------|
| <i>host</i> | host handler |
|-------------|--------------|

**74.6.4.5.21 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. |

**74.6.4.5.22 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 \* cache line size.

## Parameters

|                |                   |
|----------------|-------------------|
| <i>host</i>    | host handler      |
| <i>content</i> | transfer content. |

**74.6.4.5.23** `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. |

**74.6.4.5.24** `void SDMMCHOST_Reset ( sdmmchost_t * host )`

## Parameters

|             |              |
|-------------|--------------|
| <i>host</i> | host handler |
|-------------|--------------|

**74.6.4.5.25** `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.       |



## 74.7 SDMMC Common

### 74.7.1 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\\_BYTE\\_SEQUENCE](#)(x) ([\\_\\_REV](#)(x))  
*Reverse byte sequence in uint32\_t.*
- #define [SWAP\\_HALF\\_WROD\\_BYTE\\_SEQUENCE](#)(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\_BYTE  
*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** (400000U)  
*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)

- *The bit mask for VOLTAGE ACCEPTED field in R7.*  
• #define **SDSPI\_R7\_VOLTAGE\_27\_36\_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\_FUNCN** (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 field 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 field 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 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\_SDMMC,

```

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\_IOVoltageCtrlByHost = 1U,  
     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\_R1IdleStateFlag = (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_ApplicationSetBusWidth = 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\_FunctionSDR12Default = 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 `sddmmc_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` }
- MMC card classified as density level.*
- 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 mdde 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) }

- mmc extended csd flags*
- enum `mmc_boot_mode_t` {
`kMMC_BootModeNormal` = 0U,
`kMMC_BootModeAlternative` = 1U }
MMC card boot mode.

common function

tuning pattern

- `status_t SDMMC_SelectCard` (`sdmmcchost_t` *host, uint32_t relativeAddress, bool isSelected)
Selects the card to put it into transfer state.
- `status_t SDMMC_SendApplicationCommand` (`sdmmcchost_t` *host, uint32_t relativeAddress)
Sends an application command.
- `status_t SDMMC_SetBlockCount` (`sdmmcchost_t` *host, uint32_t blockCount)
Sets the block count.
- `status_t SDMMC_GoIdle` (`sdmmcchost_t` *host)
Sets the card to be idle state.
- `status_t SDMMC_SetBlockSize` (`sdmmcchost_t` *host, uint32_t blockSize)
Sets data block size.
- `status_t SDMMC_SetCardInactive` (`sdmmcchost_t` *host)
Sets card to inactive status.

74.7.2 Data Structure Documentation

74.7.2.1 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

74.7.2.2 struct sd_io_voltage_t

Data Fields

- `sd_io_voltage_ctrl_type_t` type

- *io voltage switch type*
• `sd_io_voltage_func_t func`
io voltage switch function

74.7.2.3 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

74.7.2.4 struct sdio_card_int_t

Data Fields

- `void * userData`
user data
- `sdio_int_t cardInterrupt`
card int call back

74.7.2.5 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

74.7.2.6 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

74.7.2.7 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

74.7.2.8 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 operation 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 enable 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

74.7.2.9 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

74.7.2.10 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].

74.7.2.11 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`
*Data read access-time-2 in clock cycles (NSAC*100) [111:104].*
- `uint16_t cardCommandClass`
Maximum data transfer rate [103:96].
- `uint8_t readBlockLength`
Card command classes [95:84].
- `uint16_t flags`
Maximum read data block length [83:80].
- `uint32_t deviceSize`
Flags in `_sd_csd_flag`.
- `uint8_t readCurrentVddMin`
Device size [73:62].
- `uint8_t readCurrentVddMax`
Maximum read current at VDD min [61:59].
- `uint8_t writeCurrentVddMin`
Maximum read current at VDD max [58:56].
- `uint8_t writeCurrentVddMax`
Maximum write current at VDD min [55:53].
- `uint8_t deviceSizeMultiplier`
Maximum write current at VDD max [52:50].
- `uint8_t eraseSectorSize`
Device size multiplier [49:47].
- `uint8_t writeProtectGroupSize`
Erase sector size [45:39].
- `uint8_t writeSpeedFactor`
Write protect group size [38:32].
- `uint8_t writeBlockLength`
Write speed factor [28:26].
- `uint8_t fileFormat`
Maximum write data block length [25:22].
- `uint8_t fileFormat`
File format [11:10].

74.7.2.12 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]

74.7.2.13 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.

74.7.2.14 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](#)

- *Max.*
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]

74.7.2.15 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 [extendedCsdVersion](#)
 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]

74.7.2.16 `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.

74.7.2.17 `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

74.7.3 Macro Definition Documentation

74.7.3.1 #define SDMMC_LOG(*format*, ...)

74.7.3.2 #define READ_MMC_TRANSFER_SPEED_FREQUENCY_UNIT(*CSD*) (((CSD->transferSpeed) & MMC_TRANSFER_SPEED_FREQUENCY_UNIT_MASK) >> MMC_TRANSFER_SPEED_FREQUENCY_UNIT_SHIFT)

74.7.3.3 #define READ_MMC_TRANSFER_SPEED_MULTIPLIER(*CSD*) (((CSD->transferSpeed) & MMC_TRANSFER_SPEED_MULTIPLIER_MASK) >> MMC_TRANSFER_SPEED_MULTIPLIER_SHIFT)

74.7.3.4 #define MMC_EXTENDED_CSD_BYTES (512U)

74.7.3.5 #define SD_PRODUCT_NAME_BYTES (5U)

74.7.3.6 #define MMC_PRODUCT_NAME_BYTES (6U)

74.7.3.7 #define MMC_SWITCH_COMMAND_SET_SHIFT (0U)

74.7.3.8 #define MMC_SWITCH_COMMAND_SET_MASK (0x00000007U)

74.7.4 Enumeration Type Documentation

74.7.4.1 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 invalid voltage
kStatus_SDMMC_SDIO_SwitchHighSpeedFail switch to high speed fail
kStatus_SDMMC_SDIO_ReadCISFail read CIS fail
kStatus_SDMMC_SDIO_InvalidCard invalid SDIO card
kStatus_SDMMC_TuningFail tuning fail
kStatus_SDMMC_SwitchVoltageFail switch voltage fail
kStatus_SDMMC_SwitchVoltage18VFail3VSuccess 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

74.7.4.2 anonymous enum

Enumerator

kSDMMC_SignalLineCmd cmd line
kSDMMC_SignalLineData0 data line
kSDMMC_SignalLineData1 data line
kSDMMC_SignalLineData2 data line
kSDMMC_SignalLineData3 data line
kSDMMC_SignalLineData4 data line
kSDMMC_SignalLineData5 data line

kSDMMC_SignalLineData6 data line

kSDMMC_SignalLineData7 data line

74.7.4.3 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

74.7.4.4 anonymous enum

Enumerator

kSDMMC_BusWdith1Bit card bus 1 width

kSDMMC_BusWdith4Bit card bus 4 width

kSDMMC_BusWdith8Bit card bus 8 width

74.7.4.5 anonymous enum

Enumerator

kSDMMC_Support8BitWidth 8 bit data width capability

74.7.4.6 anonymous enum

Enumerator

kSDMMC_DataPacketFormatLSBFirst usual data packet format LSB first, MSB last

kSDMMC_DataPacketFormatMSBFirst Wide width data packet format MSB first, LSB last.

74.7.4.7 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

74.7.4.8 anonymous enum

Enumerator

kSD_Inserted card is inserted
kSD_Removed card is removed

74.7.4.9 anonymous enum

Enumerator

kSD_DAT3PullDown data3 pull down
kSD_DAT3PullUp data3 pull up

74.7.4.10 enum sd_io_voltage_ctrl_type_t

Enumerator

kSD_IOVoltageCtrlNotSupport io voltage control not support
kSD_IOVoltageCtrlByHost io voltage control by host
kSD_IOVoltageCtrlByGpio io voltage control by gpio

74.7.4.11 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

74.7.4.12 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.

74.7.4.13 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.

74.7.4.14 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.

74.7.4.15 anonymous enum

Enumerator

kSDSPI_DataErrorTokenError Data error.
kSDSPI_DataErrorTokenCardControllerError Card controller error.
kSDSPI_DataErrorTokenCardEccFailed Card ecc error.
kSDSPI_DataErrorTokenOutOfRange Out of range.

74.7.4.16 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.

74.7.4.17 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.

74.7.4.18 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.

74.7.4.19 enum sdspi_command_t

Enumerator

kSDSPI_CommandCrc Command crc protection on/off.

74.7.4.20 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 Connect/Disconnect pull up on detect pin.
kSD_ApplicationSendScr Send Scr.

74.7.4.21 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.

74.7.4.22 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.

74.7.4.23 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.

74.7.4.24 enum sd_switch_mode_t

Enumerator

kSD_SwitchCheck SD switch mode 0: check function.
kSD_SwitchSet SD switch mode 1: set function.

74.7.4.25 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].

74.7.4.26 anonymous enum

Enumerator

kSD_ScrDataStatusAfterErase Data status after erases [55:55].
kSD_ScrSdSpecification3 Specification version 3.00 or higher [47:47].

74.7.4.27 anonymous enum

Enumerator

kSD_FunctionSDR12Deafult SDR12 mode & default.
kSD_FunctionSDR25HighSpeed SDR25 & high speed.

kSD_FunctionSDR50 SDR50 mode.
kSD_FunctionSDR104 SDR104 mode.
kSD_FunctionDDR50 DDR50 mode.

74.7.4.28 anonymous enum

Enumerator

kSD_GroupTimingMode access mode group
kSD_GroupCommandSystem command system group
kSD_GroupDriverStrength driver strength group
kSD_GroupCurrentLimit current limit group

74.7.4.29 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.

74.7.4.30 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

74.7.4.31 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

74.7.4.32 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.

74.7.4.33 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.

74.7.4.34 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

74.7.4.35 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

74.7.4.36 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

74.7.4.37 anonymous enum

Enumerator

kSDIO_OcrPowerUpBusyFlag Power up busy status.
kSDIO_OcrIONumber number of IO function
kSDIO_OcrMemPresent memory present flag
kSDIO_OcrVdd20_21Flag VDD 2.0-2.1.
kSDIO_OcrVdd21_22Flag VDD 2.1-2.2.
kSDIO_OcrVdd22_23Flag VDD 2.2-2.3.
kSDIO_OcrVdd23_24Flag VDD 2.3-2.4.
kSDIO_OcrVdd24_25Flag VDD 2.4-2.5.
kSDIO_OcrVdd25_26Flag VDD 2.5-2.6.
kSDIO_OcrVdd26_27Flag VDD 2.6-2.7.
kSDIO_OcrVdd27_28Flag VDD 2.7-2.8.
kSDIO_OcrVdd28_29Flag VDD 2.8-2.9.
kSDIO_OcrVdd29_30Flag VDD 2.9-3.0.
kSDIO_OcrVdd30_31Flag VDD 2.9-3.0.
kSDIO_OcrVdd31_32Flag VDD 3.0-3.1.
kSDIO_OcrVdd32_33Flag VDD 3.1-3.2.
kSDIO_OcrVdd33_34Flag VDD 3.2-3.3.
kSDIO_OcrVdd34_35Flag VDD 3.3-3.4.
kSDIO_OcrVdd35_36Flag VDD 3.4-3.5.

74.7.4.38 anonymous enum

Enumerator

kSDIO_CCCRSupportDirectCmdDuringDataTrans support direct cmd during data transfer
kSDIO_CCCRSupportMultiBlock support multi block mode
kSDIO_CCCRSupportReadWait support read wait
kSDIO_CCCRSupportSuspendResume support suspend resume
kSDIO_CCCRSupportIntDuring4BitDataTrans support interrupt during 4-bit data transfer
kSDIO_CCCRSupportLowSpeed1Bit support low speed 1bit mode
kSDIO_CCCRSupportLowSpeed4Bit support low speed 4bit mode
kSDIO_CCCRSupportMasterPowerControl support master power control
kSDIO_CCCRSupportHighSpeed support high speed
kSDIO_CCCRSupportContinuousSPIInt support continuous SPI interrupt

74.7.4.39 anonymous enum

Enumerator

kSDIO_FBRSupportCSA function support CSA
kSDIO_FBRSupportPowerSelection function support power selection

74.7.4.40 enum sdio_bus_width_t

Enumerator

kSDIO_DataBus1Bit 1 bit bus mode
kSDIO_DataBus4Bit 4 bit bus mode
kSDIO_DataBus8Bit 8 bit bus mode

74.7.4.41 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.

74.7.4.42 enum mmc_classified_voltage_t

Enumerator

kMMC_ClassifiedVoltageHigh High-voltage MMC card.
kMMC_ClassifiedVoltageDual Dual-voltage MMC card.

74.7.4.43 enum mmc_classified_density_t

Enumerator

kMMC_ClassifiedDensityWithin2GB Density byte is less than or equal 2GB.

74.7.4.44 enum mmc_access_mode_t

Enumerator

- kMMC_AccessModeByte* The card should be accessed as byte.
kMMC_AccessModeSector The card should be accessed as sector.

74.7.4.45 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.

74.7.4.46 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.

74.7.4.47 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.

74.7.4.48 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.

74.7.4.49 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.

74.7.4.50 anonymous enum

Enumerator

kMMC_SupportAlternateBoot support alternative boot mode

kMMC_SupportDDRBoot support DDR boot mode

kMMC_SupportHighSpeedBoot support high speed boot mode

74.7.4.51 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.

74.7.4.52 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.

74.7.4.53 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.

74.7.4.54 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 data rate

74.7.4.55 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

74.7.4.56 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

74.7.4.57 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

74.7.4.58 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.

74.7.4.59 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.

74.7.4.60 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.

74.7.4.61 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.

74.7.4.62 enum mmc_extended_csd_flags_t

Enumerator

kMMC_ExtCsdExtPartitionSupport partitioning support[160]
kMMC_ExtCsdEnhancePartitionSupport partitioning support[160]
kMMC_ExtCsdPartitioningSupport partitioning support[160]
kMMC_ExtCsdPrgCIDCSDInDDRModesSupport 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]

74.7.4.63 enum mmc_boot_mode_t

Enumerator

kMMC_BootModeNormal Normal boot.
kMMC_BootModeAlternative Alternative boot.

74.7.5 Function Documentation

74.7.5.1 `status_t SDMMC_SelectCard (sdmmcchost_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.

74.7.5.2 **status_t SDMMC_SendApplicationCommand (sdmmc_host_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_Card-NotSupport</i>	Card doesn't support.
<i>kStatus_Success</i>	Operate successfully.

74.7.5.3 **status_t SDMMC_SetBlockCount (sdmmc_host_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.

74.7.5.4 status_t SDMMC_Goldle (sdmmchost_t * host)

Parameters

<i>host</i>	host handler.
-------------	---------------

Return values

<i>kStatus_SDMMC_-TransferFailed</i>	Transfer failed.
<i>kStatus_Success</i>	Operate successfully.

74.7.5.5 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.

74.7.5.6 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 75

CODEC Driver

75.1 Overview

The MCUXpresso SDK provides a codec abstraction driver interface to access codec register.

Modules

- [CODEC Common Driver](#)
- [CODEC I2C Driver](#)
- [WM8960 Driver](#)

75.2 CODEC Common Driver

75.2.1 Overview

The codec common driver provides a codec control abstraction interface.

Modules

- [WM8960 Adapter](#)

Data Structures

- struct [codec_config_t](#)
Initialize structure of the codec. [More...](#)
- struct [codec_capability_t](#)
codec capability [More...](#)
- struct [codec_handle_t](#)
Codec handle definition. [More...](#)

Macros

- #define [CODEC_VOLUME_MAX_VALUE](#) (100U)
codec maximum volume range

Enumerations

- enum {
 [kStatus_CODEC_NotSupport](#) = MAKE_STATUS(kStatusGroup_CODEC, 0U),
 [kStatus_CODEC_DeviceNotRegistered](#) = MAKE_STATUS(kStatusGroup_CODEC, 1U),
 [kStatus_CODEC_I2CBusInitialFailed](#),
 [kStatus_CODEC_I2CCommandTransferFailed](#) }
 CODEC status.
- enum [codec_audio_protocol_t](#) {
 [kCODEC_BusI2S](#) = 0U,
 [kCODEC_BusLeftJustified](#) = 1U,
 [kCODEC_BusRightJustified](#) = 2U,
 [kCODEC_BusPCMA](#) = 3U,
 [kCODEC_BusPCMB](#) = 4U,
 [kCODEC_BusTDM](#) = 5U }
 AUDIO format definition.
- enum {


```

kCODEC_AudioSampleRate8KHz = 8000U,
kCODEC_AudioSampleRate11025Hz = 11025U,
kCODEC_AudioSampleRate12KHz = 12000U,
kCODEC_AudioSampleRate16KHz = 16000U,
kCODEC_AudioSampleRate22050Hz = 22050U,
kCODEC_AudioSampleRate24KHz = 24000U,
kCODEC_AudioSampleRate32KHz = 32000U,
kCODEC_AudioSampleRate44100Hz = 44100U,
kCODEC_AudioSampleRate48KHz = 48000U,
kCODEC_AudioSampleRate96KHz = 96000U,
kCODEC_AudioSampleRate192KHz = 192000U,
kCODEC_AudioSampleRate384KHz = 384000U }

```

audio sample rate definition

- enum {


```

kCODEC_AudioBitWidth16bit = 16U,
kCODEC_AudioBitWidth20bit = 20U,
kCODEC_AudioBitWidth24bit = 24U,
kCODEC_AudioBitWidth32bit = 32U }

```

audio bit width

- enum codec_module_t {


```

kCODEC_ModuleADC = 0U,
kCODEC_ModuleDAC = 1U,
kCODEC_ModulePGA = 2U,
kCODEC_ModuleHeadphone = 3U,
kCODEC_ModuleSpeaker = 4U,
kCODEC_ModuleLinein = 5U,
kCODEC_ModuleLineout = 6U,
kCODEC_ModuleVref = 7U,
kCODEC_ModuleMicbias = 8U,
kCODEC_ModuleMic = 9U,
kCODEC_ModuleI2SIn = 10U,
kCODEC_ModuleI2SOut = 11U,
kCODEC_ModuleMixer = 12U }

```

audio codec module

- enum codec_module_ctrl_cmd_t { kCODEC_ModuleSwitchI2SInInterface = 0U }

audio codec module control cmd

- enum {


```

kCODEC_ModuleI2SInInterfacePCM = 0U,
kCODEC_ModuleI2SInInterfaceDSD = 1U }

```

audio codec module digital interface

- enum {


```

kCODEC_RecordSourceDifferentialLine = 1U,
kCODEC_RecordSourceLineInput = 2U,
kCODEC_RecordSourceDifferentialMic = 4U,
kCODEC_RecordSourceDigitalMic = 8U,
kCODEC_RecordSourceSingleEndMic = 16U }

```

- audio codec module record source value*
 - enum {
 - kCODEC_RecordChannelLeft1 = 1U,
 - kCODEC_RecordChannelLeft2 = 2U,
 - kCODEC_RecordChannelLeft3 = 4U,
 - kCODEC_RecordChannelRight1 = 1U,
 - kCODEC_RecordChannelRight2 = 2U,
 - kCODEC_RecordChannelRight3 = 4U,
 - kCODEC_RecordChannelDifferentialPositive1 = 1U,
 - kCODEC_RecordChannelDifferentialPositive2 = 2U,
 - kCODEC_RecordChannelDifferentialPositive3 = 4U,
 - kCODEC_RecordChannelDifferentialNegative1 = 8U,
 - kCODEC_RecordChannelDifferentialNegative2 = 16U,
 - kCODEC_RecordChannelDifferentialNegative3 = 32U }
 - audio codec record channel*
 - enum {
 - kCODEC_PlaySourcePGA = 1U,
 - kCODEC_PlaySourceInput = 2U,
 - kCODEC_PlaySourceDAC = 4U,
 - kCODEC_PlaySourceMixerIn = 1U,
 - kCODEC_PlaySourceMixerInLeft = 2U,
 - kCODEC_PlaySourceMixerInRight = 4U,
 - kCODEC_PlaySourceAux = 8U }
 - audio codec module play source value*
 - enum {
 - kCODEC_PlayChannelHeadphoneLeft = 1U,
 - kCODEC_PlayChannelHeadphoneRight = 2U,
 - kCODEC_PlayChannelSpeakerLeft = 4U,
 - kCODEC_PlayChannelSpeakerRight = 8U,
 - kCODEC_PlayChannelLineOutLeft = 16U,
 - kCODEC_PlayChannelLineOutRight = 32U,
 - kCODEC_PlayChannelLeft0 = 1U,
 - kCODEC_PlayChannelRight0 = 2U,
 - kCODEC_PlayChannelLeft1 = 4U,
 - kCODEC_PlayChannelRight1 = 8U,
 - kCODEC_PlayChannelLeft2 = 16U,
 - kCODEC_PlayChannelRight2 = 32U,
 - kCODEC_PlayChannelLeft3 = 64U,
 - kCODEC_PlayChannelRight3 = 128U }
 - codec play channel*
 - enum {

```
kCODEC_VolumeHeadphoneLeft = 1U,  
kCODEC_VolumeHeadphoneRight = 2U,  
kCODEC_VolumeSpeakerLeft = 4U,  
kCODEC_VolumeSpeakerRight = 8U,  
kCODEC_VolumeLineOutLeft = 16U,  
kCODEC_VolumeLineOutRight = 32U,  
kCODEC_VolumeLeft0 = 1UL << 0U,  
kCODEC_VolumeRight0 = 1UL << 1U,  
kCODEC_VolumeLeft1 = 1UL << 2U,  
kCODEC_VolumeRight1 = 1UL << 3U,  
kCODEC_VolumeLeft2 = 1UL << 4U,  
kCODEC_VolumeRight2 = 1UL << 5U,  
kCODEC_VolumeLeft3 = 1UL << 6U,  
kCODEC_VolumeRight3 = 1UL << 7U,  
kCODEC_VolumeDAC = 1UL << 8U }
```

codec volume setting

- enum {

```

kCODEC_SupportModuleADC = 1U << 0U,
kCODEC_SupportModuleDAC = 1U << 1U,
kCODEC_SupportModulePGA = 1U << 2U,
kCODEC_SupportModuleHeadphone = 1U << 3U,
kCODEC_SupportModuleSpeaker = 1U << 4U,
kCODEC_SupportModuleLinein = 1U << 5U,
kCODEC_SupportModuleLineout = 1U << 6U,
kCODEC_SupportModuleVref = 1U << 7U,
kCODEC_SupportModuleMicbias = 1U << 8U,
kCODEC_SupportModuleMic = 1U << 9U,
kCODEC_SupportModuleI2SIn = 1U << 10U,
kCODEC_SupportModuleI2SOut = 1U << 11U,
kCODEC_SupportModuleMixer = 1U << 12U,
kCODEC_SupportModuleI2SInSwitchInterface = 1U << 13U,
kCODEC_SupportPlayChannelLeft0 = 1U << 0U,
kCODEC_SupportPlayChannelRight0 = 1U << 1U,
kCODEC_SupportPlayChannelLeft1 = 1U << 2U,
kCODEC_SupportPlayChannelRight1 = 1U << 3U,
kCODEC_SupportPlayChannelLeft2 = 1U << 4U,
kCODEC_SupportPlayChannelRight2 = 1U << 5U,
kCODEC_SupportPlayChannelLeft3 = 1U << 6U,
kCODEC_SupportPlayChannelRight3 = 1U << 7U,
kCODEC_SupportPlaySourcePGA = 1U << 8U,
kCODEC_SupportPlaySourceInput = 1U << 9U,
kCODEC_SupportPlaySourceDAC = 1U << 10U,
kCODEC_SupportPlaySourceMixerIn = 1U << 11U,
kCODEC_SupportPlaySourceMixerInLeft = 1U << 12U,
kCODEC_SupportPlaySourceMixerInRight = 1U << 13U,
kCODEC_SupportPlaySourceAux = 1U << 14U,
kCODEC_SupportRecordSourceDifferentialLine = 1U << 0U,
kCODEC_SupportRecordSourceLineInput = 1U << 1U,
kCODEC_SupportRecordSourceDifferentialMic = 1U << 2U,
kCODEC_SupportRecordSourceDigitalMic = 1U << 3U,
kCODEC_SupportRecordSourceSingleEndMic = 1U << 4U,
kCODEC_SupportRecordChannelLeft1 = 1U << 6U,
kCODEC_SupportRecordChannelLeft2 = 1U << 7U,
kCODEC_SupportRecordChannelLeft3 = 1U << 8U,
kCODEC_SupportRecordChannelRight1 = 1U << 9U,
kCODEC_SupportRecordChannelRight2 = 1U << 10U,
kCODEC_SupportRecordChannelRight3 = 1U << 11U }

```

audio codec capability

Functions

- [status_t CODEC_Init](#) (codec_handle_t *handle, [codec_config_t](#) *config)
Codec initialization.
- [status_t CODEC_Deinit](#) (codec_handle_t *handle)
Codec de-initialization.
- [status_t CODEC_SetFormat](#) (codec_handle_t *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- [status_t CODEC_ModuleControl](#) (codec_handle_t *handle, [codec_module_ctrl_cmd_t](#) cmd, uint32_t data)
codec module control.
- [status_t CODEC_SetVolume](#) (codec_handle_t *handle, uint32_t channel, uint32_t volume)
set audio codec pl volume.
- [status_t CODEC_SetMute](#) (codec_handle_t *handle, uint32_t channel, bool mute)
set audio codec module mute.
- [status_t CODEC_SetPower](#) (codec_handle_t *handle, [codec_module_t](#) module, bool powerOn)
set audio codec power.
- [status_t CODEC_SetRecord](#) (codec_handle_t *handle, uint32_t recordSource)
codec set record source.
- [status_t CODEC_SetRecordChannel](#) (codec_handle_t *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- [status_t CODEC_SetPlay](#) (codec_handle_t *handle, uint32_t playSource)
codec set play source.

Driver version

- #define [FSL_CODEC_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 3, 0))
CLOCK driver version 2.3.0.

75.2.2 Data Structure Documentation

75.2.2.1 struct codec_config_t

Data Fields

- uint32_t [codecDevType](#)
codec type
- void * [codecDevConfig](#)
Codec device specific configuration.

75.2.2.2 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

75.2.2.3 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

75.2.3 Macro Definition Documentation

75.2.3.1 #define FSL_CODEC_DRIVER_VERSION (MAKE_VERSION(2, 3, 0))

75.2.4 Enumeration Type Documentation

75.2.4.1 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.

75.2.4.2 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.

75.2.4.3 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.

75.2.4.4 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

75.2.4.5 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

75.2.4.6 enum codec_module_ctrl_cmd_t

Enumerator

kCODEC_ModuleSwitchI2SInInterface module digital interface swtch.

75.2.4.7 anonymous enum

Enumerator

kCODEC_ModuleI2SInInterfacePCM Pcm interface.
kCODEC_ModuleI2SInInterfaceDSD DSD interface.

75.2.4.8 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

75.2.4.9 anonymous enum

Enumerator

kCODEC_RecordChannelLeft1 left record channel 1
kCODEC_RecordChannelLeft2 left record channel 2
kCODEC_RecordChannelLeft3 left record channel 3
kCODEC_RecordChannelRight1 right record channel 1
kCODEC_RecordChannelRight2 right record channel 2
kCODEC_RecordChannelRight3 right record channel 3
kCODEC_RecordChannelDifferentialPositive1 differential positive record channel 1

<i>kCODEC_RecordChannelDifferentialPositive2</i>	differential positive record channel 2
<i>kCODEC_RecordChannelDifferentialPositive3</i>	differential positive record channel 3
<i>kCODEC_RecordChannelDifferentialNegative1</i>	differential negative record channel 1
<i>kCODEC_RecordChannelDifferentialNegative2</i>	differential negative record channel 2
<i>kCODEC_RecordChannelDifferentialNegative3</i>	differential negative record channel 3

75.2.4.10 anonymous enum

Enumerator

<i>kCODEC_PlaySourcePGA</i>	play source PGA, bypass ADC
<i>kCODEC_PlaySourceInput</i>	play source Input3
<i>kCODEC_PlaySourceDAC</i>	play source DAC
<i>kCODEC_PlaySourceMixerIn</i>	play source mixer in
<i>kCODEC_PlaySourceMixerInLeft</i>	play source mixer in left
<i>kCODEC_PlaySourceMixerInRight</i>	play source mixer in right
<i>kCODEC_PlaySourceAux</i>	play source mixer in AUx

75.2.4.11 anonymous enum

Enumerator

<i>kCODEC_PlayChannelHeadphoneLeft</i>	play channel headphone left
<i>kCODEC_PlayChannelHeadphoneRight</i>	play channel headphone right
<i>kCODEC_PlayChannelSpeakerLeft</i>	play channel speaker left
<i>kCODEC_PlayChannelSpeakerRight</i>	play channel speaker right
<i>kCODEC_PlayChannelLineOutLeft</i>	play channel lineout left
<i>kCODEC_PlayChannelLineOutRight</i>	play channel lineout right
<i>kCODEC_PlayChannelLeft0</i>	play channel left0
<i>kCODEC_PlayChannelRight0</i>	play channel right0
<i>kCODEC_PlayChannelLeft1</i>	play channel left1
<i>kCODEC_PlayChannelRight1</i>	play channel right1
<i>kCODEC_PlayChannelLeft2</i>	play channel left2
<i>kCODEC_PlayChannelRight2</i>	play channel right2
<i>kCODEC_PlayChannelLeft3</i>	play channel left3
<i>kCODEC_PlayChannelRight3</i>	play channel right3

75.2.4.12 anonymous enum

Enumerator

<i>kCODEC_VolumeHeadphoneLeft</i>	headphone left volume
<i>kCODEC_VolumeHeadphoneRight</i>	headphone right volume
<i>kCODEC_VolumeSpeakerLeft</i>	speaker left volume
<i>kCODEC_VolumeSpeakerRight</i>	speaker right volume

kCODEC_VolumeLineOutLeft lineout left volume
kCODEC_VolumeLineOutRight lineout right volume
kCODEC_VolumeLeft0 left0 volume
kCODEC_VolumeRight0 right0 volume
kCODEC_VolumeLeft1 left1 volume
kCODEC_VolumeRight1 right1 volume
kCODEC_VolumeLeft2 left2 volume
kCODEC_VolumeRight2 right2 volume
kCODEC_VolumeLeft3 left3 volume
kCODEC_VolumeRight3 right3 volume
kCODEC_VolumeDAC dac volume

75.2.4.13 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

75.2.5 Function Documentation

75.2.5.1 `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.

75.2.5.2 `status_t CODEC_Deinit (codec_handle_t * handle)`

Parameters

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

Returns

kStatus_Success is success, else de-initial failed.

75.2.5.3 `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.

75.2.5.4 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.

75.2.5.5 status_t CODEC_SetVolume (codec_handle_t * *handle*, uint32_t *channel*, uint32_t *volume*)

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec volume channel, can be a value or combine value of <code>_codec_volume_capability</code> or <code>_codec_play_channel</code> .
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

`kStatus_Success` is success, else configure failed.

75.2.5.6 `status_t CODEC_SetMute (codec_handle_t * handle, uint32_t channel, bool mute)`

Parameters

<i>handle</i>	codec handle.
<i>channel</i>	audio codec volume channel, can be a value or combine value of <code>_codec_volume_capability</code> or <code>_codec_play_channel</code> .
<i>mute</i>	true is mute, false is unmute.

Returns

`kStatus_Success` is success, else configure failed.

75.2.5.7 `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.

75.2.5.8 `status_t CODEC_SetRecord (codec_handle_t * handle, uint32_t recordSource)`

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of <code>_codec_record_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

75.2.5.9 `status_t CODEC_SetRecordChannel (codec_handle_t * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)`

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference <code>_codec_record_channel</code> , can be a value combine of member in <code>_codec_record_channel</code> .
<i>rightRecord-Channel</i>	audio codec record channel, reference <code>_codec_record_channel</code> , can be a value combine of member in <code>_codec_record_channel</code> .

Returns

`kStatus_Success` is success, else configure failed.

75.2.5.10 `status_t CODEC_SetPlay (codec_handle_t * handle, uint32_t playSource)`

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of <code>_codec_play_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

75.3 CODEC I2C Driver

75.3.1 Overview

The codec common driver provides a codec control abstraction interface.

Data Structures

- struct `codec_i2c_config_t`
CODEC I2C configurations structure. [More...](#)

Macros

- #define `CODEC_I2C_MASTER_HANDLER_SIZE` `HAL_I2C_MASTER_HANDLE_SIZE`
codec i2c handler

Enumerations

- enum `codec_reg_addr_t` {
 `kCODEC_RegAddr8Bit` = 1U,
 `kCODEC_RegAddr16Bit` = 2U }
CODEC device register address type.
- enum `codec_reg_width_t` {
 `kCODEC_RegWidth8Bit` = 1U,
 `kCODEC_RegWidth16Bit` = 2U,
 `kCODEC_RegWidth32Bit` = 4U }
CODEC device register width.

Functions

- `status_t CODEC_I2C_Init` (void *handle, uint32_t i2cInstance, uint32_t i2cBaudrate, uint32_t i2cSourceClockHz)
Codec i2c bus initialization.
- `status_t CODEC_I2C_Deinit` (void *handle)
Codec i2c de-initialization.
- `status_t CODEC_I2C_Send` (void *handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t *txBuff, uint8_t txBuffSize)
codec i2c send function.
- `status_t CODEC_I2C_Receive` (void *handle, uint8_t deviceAddress, uint32_t subAddress, uint8_t subaddressSize, uint8_t *rxBuff, uint8_t rxBuffSize)
codec i2c receive function.

75.3.2 Data Structure Documentation

75.3.2.1 struct codec_i2c_config_t

Data Fields

- uint32_t [codecI2CInstance](#)
i2c bus instance
- uint32_t [codecI2CSourceClock](#)
i2c bus source clock frequency

75.3.3 Enumeration Type Documentation

75.3.3.1 enum codec_reg_addr_t

Enumerator

- kCODEC_RegAddr8Bit*** 8-bit register address.
kCODEC_RegAddr16Bit 16-bit register address.

75.3.3.2 enum codec_reg_width_t

Enumerator

- kCODEC_RegWidth8Bit*** 8-bit register width.
kCODEC_RegWidth16Bit 16-bit register width.
kCODEC_RegWidth32Bit 32-bit register width.

75.3.4 Function Documentation

75.3.4.1 status_t CODEC_I2C_Init (void * *handle*, uint32_t *i2cInstance*, uint32_t *i2cBaudrate*, uint32_t *i2cSourceClockHz*)

Parameters

<i>handle</i>	i2c master handle.
<i>i2cInstance</i>	instance number of the i2c bus, such as 0 is corresponding to I2C0.

<i>i2cBaudrate</i>	i2c baudrate.
<i>i2cSource-ClockHz</i>	i2c source clock frequency.

Returns

kStatus_HAL_I2cSuccess is success, else initial failed.

75.3.4.2 status_t CODEC_I2C_Deinit (void * *handle*)

Parameters

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

Returns

kStatus_HAL_I2cSuccess is success, else deinitial failed.

75.3.4.3 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.

75.3.4.4 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.

75.4 WM8960 Driver

75.4.1 Overview

The wm8960 driver provides a codec control interface.

Data Structures

- struct [wm8960_audio_format_t](#)
wm8960 audio format [More...](#)
- struct [wm8960_master_sysclk_config_t](#)
wm8960 master system clock configuration [More...](#)
- struct [wm8960_config_t](#)
Initialize structure of WM8960. [More...](#)
- struct [wm8960_handle_t](#)
wm8960 codec handler [More...](#)

Macros

- #define [WM8960_I2C_HANDLER_SIZE](#) CODEC_I2C_MASTER_HANDLER_SIZE
wm8960 handle size
- #define [WM8960_LINVOL](#) 0x0U
Define the register address of WM8960.
- #define [WM8960_CACHEREGNUM](#) 56U
Cache register number.
- #define [WM8960_CLOCK2_BCLK_DIV_MASK](#) 0xFU
WM8960 CLOCK2 bits.
- #define [WM8960_IFACE1_FORMAT_MASK](#) 0x03U
WM8960_IFACE1 FORMAT bits.
- #define [WM8960_IFACE1_WL_MASK](#) 0x0CU
WM8960_IFACE1 WL bits.
- #define [WM8960_IFACE1_LRP_MASK](#) 0x10U
WM8960_IFACE1 LRP bit.
- #define [WM8960_IFACE1_DLR_SWAP_MASK](#) 0x20U
WM8960_IFACE1 DLR_SWAP bit.
- #define [WM8960_IFACE1_MS_MASK](#) 0x40U
WM8960_IFACE1 MS bit.
- #define [WM8960_IFACE1_BCLK_INV_MASK](#) 0x80U
WM8960_IFACE1 BCLK_INV bit.
- #define [WM8960_IFACE1_ALR_SWAP_MASK](#) 0x100U
WM8960_IFACE1 ALR_SWAP bit.
- #define [WM8960_POWER1_VREF_MASK](#) 0x40U
WM8960_POWER1.
- #define [WM8960_POWER2_DACL_MASK](#) 0x100U
WM8960_POWER2.
- #define [WM8960_I2C_ADDR](#) 0x1A
WM8960 I2C address.
- #define [WM8960_I2C_BAUDRATE](#) (100000U)

- *WM8960 I2C baudrate.*
- #define `WM8960_ADC_MAX_VOLUME_VALUE` 0xFFU
WM8960 maximum volume value.

Enumerations

- enum `wm8960_module_t` {
`kWM8960_ModuleADC` = 0,
`kWM8960_ModuleDAC` = 1,
`kWM8960_ModuleVREF` = 2,
`kWM8960_ModuleHP` = 3,
`kWM8960_ModuleMICB` = 4,
`kWM8960_ModuleMIC` = 5,
`kWM8960_ModuleLineIn` = 6,
`kWM8960_ModuleLineOut` = 7,
`kWM8960_ModuleSpeaker` = 8,
`kWM8960_ModuleOMIX` = 9 }
Modules in WM8960 board.
- enum {
`kWM8960_HeadphoneLeft` = 1,
`kWM8960_HeadphoneRight` = 2,
`kWM8960_SpeakerLeft` = 4,
`kWM8960_SpeakerRight` = 8 }
wm8960 play channel
- enum `wm8960_play_source_t` {
`kWM8960_PlaySourcePGA` = 1,
`kWM8960_PlaySourceInput` = 2,
`kWM8960_PlaySourceDAC` = 4 }
wm8960 play source
- enum `wm8960_route_t` {
`kWM8960_RouteBypass` = 0,
`kWM8960_RoutePlayback` = 1,
`kWM8960_RoutePlaybackandRecord` = 2,
`kWM8960_RouteRecord` = 5 }
WM8960 data route.
- enum `wm8960_protocol_t` {
`kWM8960_BusI2S` = 2,
`kWM8960_BusLeftJustified` = 1,
`kWM8960_BusRightJustified` = 0,
`kWM8960_BusPCMA` = 3,
`kWM8960_BusPCMB` = 3 | (1 << 4) }
The audio data transfer protocol choice.
- enum `wm8960_input_t` {

```

kWM8960_InputClosed = 0,
kWM8960_InputSingleEndedMic = 1,
kWM8960_InputDifferentialMicInput2 = 2,
kWM8960_InputDifferentialMicInput3 = 3,
kWM8960_InputLineINPUT2 = 4,
kWM8960_InputLineINPUT3 = 5 }
    wm8960 input source
• enum {
    kWM8960_AudioSampleRate8KHz = 8000U,
    kWM8960_AudioSampleRate11025Hz = 11025U,
    kWM8960_AudioSampleRate12KHz = 12000U,
    kWM8960_AudioSampleRate16KHz = 16000U,
    kWM8960_AudioSampleRate22050Hz = 22050U,
    kWM8960_AudioSampleRate24KHz = 24000U,
    kWM8960_AudioSampleRate32KHz = 32000U,
    kWM8960_AudioSampleRate44100Hz = 44100U,
    kWM8960_AudioSampleRate48KHz = 48000U,
    kWM8960_AudioSampleRate96KHz = 96000U,
    kWM8960_AudioSampleRate192KHz = 192000U,
    kWM8960_AudioSampleRate384KHz = 384000U }
    audio sample rate definition
• enum {
    kWM8960_AudioBitWidth16bit = 16U,
    kWM8960_AudioBitWidth20bit = 20U,
    kWM8960_AudioBitWidth24bit = 24U,
    kWM8960_AudioBitWidth32bit = 32U }
    audio bit width
• enum wm8960_sysclk_source_t {
    kWM8960_SysClkSourceMclk = 0U,
    kWM8960_SysClkSourceInternalPLL = 1U }
    wm8960 sysclk source

```

Functions

- `status_t WM8960_Init (wm8960_handle_t *handle, const wm8960_config_t *config)`
WM8960 initialize function.
- `status_t WM8960_Deinit (wm8960_handle_t *handle)`
Deinit the WM8960 codec.
- `status_t WM8960_SetDataRoute (wm8960_handle_t *handle, wm8960_route_t route)`
Set audio data route in WM8960.
- `status_t WM8960_SetLeftInput (wm8960_handle_t *handle, wm8960_input_t input)`
Set left audio input source in WM8960.
- `status_t WM8960_SetRightInput (wm8960_handle_t *handle, wm8960_input_t input)`
Set right audio input source in WM8960.
- `status_t WM8960_SetProtocol (wm8960_handle_t *handle, wm8960_protocol_t protocol)`
Set the audio transfer protocol.

- void [WM8960_SetMasterSlave](#) ([wm8960_handle_t](#) *handle, bool master)
Set WM8960 as master or slave.
- [status_t WM8960_SetVolume](#) ([wm8960_handle_t](#) *handle, [wm8960_module_t](#) module, [uint32_t](#) volume)
Set the volume of different modules in WM8960.
- [uint32_t WM8960_GetVolume](#) ([wm8960_handle_t](#) *handle, [wm8960_module_t](#) module)
Get the volume of different modules in WM8960.
- [status_t WM8960_SetMute](#) ([wm8960_handle_t](#) *handle, [wm8960_module_t](#) module, bool isEnabled)
Mute modules in WM8960.
- [status_t WM8960_SetModule](#) ([wm8960_handle_t](#) *handle, [wm8960_module_t](#) module, bool isEnabled)
Enable/disable expected devices.
- [status_t WM8960_SetPlay](#) ([wm8960_handle_t](#) *handle, [uint32_t](#) playSource)
SET the WM8960 play source.
- [status_t WM8960_ConfigDataFormat](#) ([wm8960_handle_t](#) *handle, [uint32_t](#) sysclk, [uint32_t](#) sample_rate, [uint32_t](#) bits)
Configure the data format of audio data.
- [status_t WM8960_SetJackDetect](#) ([wm8960_handle_t](#) *handle, bool isEnabled)
Enable/disable jack detect feature.
- [status_t WM8960_WriteReg](#) ([wm8960_handle_t](#) *handle, [uint8_t](#) reg, [uint16_t](#) val)
Write register to WM8960 using I2C.
- [status_t WM8960_ReadReg](#) ([uint8_t](#) reg, [uint16_t](#) *val)
Read register from WM8960 using I2C.
- [status_t WM8960_ModifyReg](#) ([wm8960_handle_t](#) *handle, [uint8_t](#) reg, [uint16_t](#) mask, [uint16_t](#) val)
Modify some bits in the register using I2C.

Driver version

- #define [FSL_WM8960_DRIVER_VERSION](#) ([MAKE_VERSION](#)(2, 2, 0))
CLOCK driver version 2.2.0.

75.4.2 Data Structure Documentation

75.4.2.1 struct [wm8960_audio_format_t](#)

Data Fields

- [uint32_t mclk_HZ](#)
master clock frequency
- [uint32_t sampleRate](#)
sample rate
- [uint32_t bitWidth](#)
bit width

75.4.2.2 struct wm8960_master_sysclk_config_t

Data Fields

- [wm8960_sysclk_source_t](#) sysclkSource
sysclk source
- [uint32_t](#) sysclkFreq
PLL output frequency value.

75.4.2.3 struct wm8960_config_t

Data Fields

- [wm8960_route_t](#) route
Audio data route.
- [wm8960_protocol_t](#) bus
Audio transfer protocol.
- [wm8960_audio_format_t](#) format
Audio format.
- [bool](#) master_slave
Master or slave.
- [wm8960_master_sysclk_config_t](#) masterClock
master clock configurations
- [bool](#) enableSpeaker
True means enable class D speaker as output, false means no.
- [wm8960_input_t](#) leftInputSource
Left input source for WM8960.
- [wm8960_input_t](#) rightInputSource
Right input source for wm8960.
- [wm8960_play_source_t](#) playSource
play source
- [uint8_t](#) slaveAddress
wm8960 device address
- [codec_i2c_config_t](#) i2cConfig
i2c configuration

Field Documentation

(1) [wm8960_route_t](#) [wm8960_config_t::route](#)

(2) [bool](#) [wm8960_config_t::master_slave](#)

75.4.2.4 struct wm8960_handle_t

Data Fields

- [const](#) [wm8960_config_t](#) * config
wm8904 config pointer
- [uint8_t](#) i2cHandle [WM8960_I2C_HANDLER_SIZE]
i2c handle

75.4.3 Macro Definition Documentation

75.4.3.1 **#define WM8960_LINVOL 0x0U**

75.4.3.2 **#define WM8960_I2C_ADDR 0x1A**

75.4.4 Enumeration Type Documentation

75.4.4.1 enum wm8960_module_t

Enumerator

kWM8960_ModuleADC ADC module in WM8960.

kWM8960_ModuleDAC DAC module in WM8960.

kWM8960_ModuleVREF VREF module.

kWM8960_ModuleHP Headphone.

kWM8960_ModuleMICB Mic bias.

kWM8960_ModuleMIC Input Mic.

kWM8960_ModuleLineIn Analog in PGA.

kWM8960_ModuleLineOut Line out module.

kWM8960_ModuleSpeaker Speaker module.

kWM8960_ModuleOMIX Output mixer.

75.4.4.2 anonymous enum

Enumerator

kWM8960_HeadphoneLeft wm8960 headphone left channel

kWM8960_HeadphoneRight wm8960 headphone right channel

kWM8960_SpeakerLeft wm8960 speaker left channel

kWM8960_SpeakerRight wm8960 speaker right channel

75.4.4.3 enum wm8960_play_source_t

Enumerator

kWM8960_PlaySourcePGA wm8960 play source PGA

kWM8960_PlaySourceInput wm8960 play source Input

kWM8960_PlaySourceDAC wm8960 play source DAC

75.4.4.4 enum wm8960_route_t

Only provide some typical data route, not all route listed. Note: Users cannot combine any routes, once a new route is set, the previous one would be replaced.

Enumerator

kWM8960_RouteBypass LINEIN->Headphone.
kWM8960_RoutePlayback I2SIN->DAC->Headphone.
kWM8960_RoutePlaybackandRecord I2SIN->DAC->Headphone, LINEIN->ADC->I2SOUT.
kWM8960_RouteRecord LINEIN->ADC->I2SOUT.

75.4.4.5 enum wm8960_protocol_t

WM8960 only supports I2S format and PCM format.

Enumerator

kWM8960_BusI2S I2S type.
kWM8960_BusLeftJustified Left justified mode.
kWM8960_BusRightJustified Right justified mode.
kWM8960_BusPCMA PCM A mode.
kWM8960_BusPCMB PCM B mode.

75.4.4.6 enum wm8960_input_t

Enumerator

kWM8960_InputClosed Input device is closed.
kWM8960_InputSingleEndedMic Input as single ended mic, only use L/RINPUT1.
kWM8960_InputDifferentialMicInput2 Input as differential mic, use L/RINPUT1 and L/RINPUT2.
kWM8960_InputDifferentialMicInput3 Input as differential mic, use L/RINPUT1 and L/RINPUT3.
kWM8960_InputLineINPUT2 Input as line input, only use L/RINPUT2.
kWM8960_InputLineINPUT3 Input as line input, only use L/RINPUT3.

75.4.4.7 anonymous enum

Enumerator

kWM8960_AudioSampleRate8KHz Sample rate 8000 Hz.
kWM8960_AudioSampleRate11025Hz Sample rate 11025 Hz.
kWM8960_AudioSampleRate12KHz Sample rate 12000 Hz.
kWM8960_AudioSampleRate16KHz Sample rate 16000 Hz.
kWM8960_AudioSampleRate22050Hz Sample rate 22050 Hz.
kWM8960_AudioSampleRate24KHz Sample rate 24000 Hz.
kWM8960_AudioSampleRate32KHz Sample rate 32000 Hz.
kWM8960_AudioSampleRate44100Hz Sample rate 44100 Hz.
kWM8960_AudioSampleRate48KHz Sample rate 48000 Hz.

kWM8960_AudioSampleRate96KHz Sample rate 96000 Hz.
kWM8960_AudioSampleRate192KHz Sample rate 192000 Hz.
kWM8960_AudioSampleRate384KHz Sample rate 384000 Hz.

75.4.4.8 anonymous enum

Enumerator

kWM8960_AudioBitWidth16bit audio bit width 16
kWM8960_AudioBitWidth20bit audio bit width 20
kWM8960_AudioBitWidth24bit audio bit width 24
kWM8960_AudioBitWidth32bit audio bit width 32

75.4.4.9 enum wm8960_sysclk_source_t

Enumerator

kWM8960_SysClkSourceMclk sysclk source from external MCLK
kWM8960_SysClkSourceInternalPLL sysclk source from internal PLL

75.4.5 Function Documentation

75.4.5.1 **status_t WM8960_Init (wm8960_handle_t * *handle*, const wm8960_config_t * *config*)**

The second parameter is NULL to WM8960 in this version. If users want to change the settings, they have to use `wm8960_write_reg()` or `wm8960_modify_reg()` to set the register value of WM8960. Note: If the `codec_config` is NULL, it would initialize WM8960 using default settings. The default setting: `codec_config->route = kWM8960_RoutePlaybackandRecord` `codec_config->bus = kWM8960_BusI2S` `codec_config->master = slave`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>config</i>	WM8960 configuration structure.

75.4.5.2 **status_t WM8960_Deinit (wm8960_handle_t * *handle*)**

This function close all modules in WM8960 to save power.

Parameters

<i>handle</i>	WM8960 handle structure pointer.
---------------	----------------------------------

75.4.5.3 **status_t WM8960_SetDataRoute (wm8960_handle_t * *handle*, wm8960_route_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>	WM8960 handle structure.
<i>route</i>	Audio data route in WM8960.

75.4.5.4 **status_t WM8960_SetLeftInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>input</i>	Audio input source.

75.4.5.5 **status_t WM8960_SetRightInput (wm8960_handle_t * *handle*, wm8960_input_t *input*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>input</i>	Audio input source.

75.4.5.6 **status_t WM8960_SetProtocol (wm8960_handle_t * *handle*, wm8960_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.

75.4.5.7 void WM8960_SetMasterSlave (wm8960_handle_t * *handle*, bool *master*)

Parameters

<i>handle</i>	WM8960 handle structure.
<i>master</i>	1 represent master, 0 represent slave.

75.4.5.8 status_t WM8960_SetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*, uint32_t *volume*)

This function would set the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Module:kWM8960_ModuleADC, volume range value: 0 is mute, 1-255 is -97db to 30db Module:kWM8960_ModuleDAC, volume range value: 0 is mute, 1-255 is -127db to 0db Module:kWM8960_ModuleHP, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db Module:kWM8960_ModuleLineIn, volume range value: 0 - 0x3F is -17.25db to 30db Module:kWM8960_ModuleSpeaker, volume range value: 0 - 2F is mute, 0x30 - 0x7F is -73db to 6db

Parameters

<i>handle</i>	WM8960 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.

75.4.5.9 uint32_t WM8960_GetVolume (wm8960_handle_t * *handle*, wm8960_module_t *module*)

This function gets the volume of WM8960 modules. Uses need to appoint the module. The function assume that left channel and right channel has the same volume.

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module to set volume, it can be ADC, DAC, Headphone and so on.

Returns

Volume value of the module.

75.4.5.10 **status_t WM8960_SetMute (wm8960_handle_t * *handle*, wm8960_module_t *module*, bool *isEnabled*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Modules need to be mute.
<i>isEnabled</i>	Mute or unmute, 1 represent mute.

75.4.5.11 **status_t WM8960_SetModule (wm8960_handle_t * *handle*, wm8960_module_t *module*, bool *isEnabled*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>module</i>	Module expected to enable.
<i>isEnabled</i>	Enable or disable moudles.

75.4.5.12 **status_t WM8960_SetPlay (wm8960_handle_t * *handle*, uint32_t *playSource*)**

Parameters

<i>handle</i>	WM8960 handle structure.
<i>playSource</i>	play source , can be a value combine of kWM8960_ModuleHeadphoneSourcePGA, kWM8960_ModuleHeadphoneSourceDAC, kWM8960_ModulePlaySourceInput, kWM8960_ModulePlayMonoRight, kWM8960_ModulePlayMonoLeft.

Returns

kStatus_WM8904_Success if successful, different code otherwise..

75.4.5.13 `status_t WM8960_ConfigDataFormat (wm8960_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>	WM8960 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 WM8960. WM8960 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 (WM8960 only supports 16bit, 20bit, 24bit and 32 bit in HW).

75.4.5.14 `status_t WM8960_SetJackDetect (wm8960_handle_t * handle, bool isEnabled)`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>isEnabled</i>	Enable or disable moudles.

75.4.5.15 `status_t WM8960_WriteReg (wm8960_handle_t * handle, uint8_t reg, uint16_t val)`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>reg</i>	The register address in WM8960.
<i>val</i>	Value needs to write into the register.

75.4.5.16 `status_t WM8960_ReadReg (uint8_t reg, uint16_t * val)`

Parameters

<i>reg</i>	The register address in WM8960.
<i>val</i>	Value written to.

75.4.5.17 `status_t WM8960_ModifyReg (wm8960_handle_t * handle, uint8_t reg, uint16_t mask, uint16_t val)`

Parameters

<i>handle</i>	WM8960 handle structure.
<i>reg</i>	The register address in WM8960.
<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.

75.4.6 WM8960 Adapter

75.4.6.1 Overview

The wm8960 adapter provides a codec unify control interface.

Macros

- #define `HAL_CODEC_WM8960_HANDLER_SIZE` (`WM8960_I2C_HANDLER_SIZE` + 4)
codec handler size

Functions

- `status_t HAL_CODEC_WM8960_Init` (void *handle, void *config)
Codec initialization.
- `status_t HAL_CODEC_WM8960_Deinit` (void *handle)
Codec de-initialization.
- `status_t HAL_CODEC_WM8960_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- `status_t HAL_CODEC_WM8960_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- `status_t HAL_CODEC_WM8960_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- `status_t HAL_CODEC_WM8960_SetPower` (void *handle, uint32_t module, bool powerOn)
set audio codec module power.
- `status_t HAL_CODEC_WM8960_SetRecord` (void *handle, uint32_t recordSource)
codec set record source.
- `status_t HAL_CODEC_WM8960_SetRecordChannel` (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
codec set record channel.
- `status_t HAL_CODEC_WM8960_SetPlay` (void *handle, uint32_t playSource)
codec set play source.
- `status_t HAL_CODEC_WM8960_ModuleControl` (void *handle, uint32_t cmd, uint32_t data)
codec module control.
- static `status_t HAL_CODEC_Init` (void *handle, void *config)
Codec initialization.
- static `status_t HAL_CODEC_Deinit` (void *handle)
Codec de-initialization.
- static `status_t HAL_CODEC_SetFormat` (void *handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth)
set audio data format.
- static `status_t HAL_CODEC_SetVolume` (void *handle, uint32_t playChannel, uint32_t volume)
set audio codec module volume.
- static `status_t HAL_CODEC_SetMute` (void *handle, uint32_t playChannel, bool isMute)
set audio codec module mute.
- static `status_t HAL_CODEC_SetPower` (void *handle, uint32_t module, bool powerOn)

- *set audio codec module power.*
static [status_t HAL_CODEC_SetRecord](#) (void *handle, uint32_t recordSource)
- *codec set record source.*
static [status_t HAL_CODEC_SetRecordChannel](#) (void *handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel)
- *codec set record channel.*
static [status_t HAL_CODEC_SetPlay](#) (void *handle, uint32_t playSource)
- *codec set play source.*
static [status_t HAL_CODEC_ModuleControl](#) (void *handle, uint32_t cmd, uint32_t data)
- *codec module control.*

75.4.6.2 Function Documentation

75.4.6.2.1 status_t HAL_CODEC_WM8960_Init (void * *handle*, void * *config*)

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

kStatus_Success is success, else initial failed.

75.4.6.2.2 status_t HAL_CODEC_WM8960_Deinit (void * *handle*)

Parameters

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

Returns

kStatus_Success is success, else de-initial failed.

75.4.6.2.3 status_t HAL_CODEC_WM8960_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.

75.4.6.2.4 status_t HAL_CODEC_WM8960_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.

75.4.6.2.5 status_t HAL_CODEC_WM8960_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.

75.4.6.2.6 status_t HAL_CODEC_WM8960_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.

75.4.6.2.7 status_t HAL_CODEC_WM8960_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.

75.4.6.2.8 status_t HAL_CODEC_WM8960_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.

75.4.6.2.9 status_t HAL_CODEC_WM8960_SetPlay (void * *handle*, uint32_t *playSource*)

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of <code>_codec_play_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

75.4.6.2.10 `status_t HAL_CODEC_WM8960_ModuleControl (void * handle, uint32_t cmd, uint32_t data)`

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <code>_codec_module_ctrl_cmd</code> .
<i>data</i>	value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations.

Returns

`kStatus_Success` is success, else configure failed.

75.4.6.2.11 `static status_t HAL_CODEC_Init (void * handle, void * config) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>config</i>	codec configuration.

Returns

`kStatus_Success` is success, else initial failed.

75.4.6.2.12 `static status_t HAL_CODEC_Deinit (void * handle) [inline], [static]`

Parameters

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

Returns

kStatus_Success is success, else de-initial failed.

75.4.6.2.13 `static status_t HAL_CODEC_SetFormat (void * handle, uint32_t mclk, uint32_t sampleRate, uint32_t bitWidth) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>mclk</i>	master clock frequency in HZ.
<i>sampleRate</i>	sample rate in HZ.
<i>bitWidth</i>	bit width.

Returns

kStatus_Success is success, else configure failed.

75.4.6.2.14 `static status_t HAL_CODEC_SetVolume (void * handle, uint32_t playChannel, uint32_t volume) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of _codec_play_channel.
<i>volume</i>	volume value, support 0 ~ 100, 0 is mute, 100 is the maximum volume value.

Returns

kStatus_Success is success, else configure failed.

75.4.6.2.15 `static status_t HAL_CODEC_SetMute (void * handle, uint32_t playChannel, bool isMute) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>playChannel</i>	audio codec play channel, can be a value or combine value of <code>_codec_play_channel</code> .
<i>isMute</i>	true is mute, false is unmute.

Returns

kStatus_Success is success, else configure failed.

75.4.6.2.16 `static status_t HAL_CODEC_SetPower (void * handle, uint32_t module, bool powerOn) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>module</i>	audio codec module.
<i>powerOn</i>	true is power on, false is power down.

Returns

kStatus_Success is success, else configure failed.

75.4.6.2.17 `static status_t HAL_CODEC_SetRecord (void * handle, uint32_t recordSource) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>recordSource</i>	audio codec record source, can be a value or combine value of <code>_codec_record_source</code> .

Returns

kStatus_Success is success, else configure failed.

75.4.6.2.18 `static status_t HAL_CODEC_SetRecordChannel (void * handle, uint32_t leftRecordChannel, uint32_t rightRecordChannel) [inline], [static]`

Parameters

<i>handle</i>	codec handle.
<i>leftRecord-Channel</i>	audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine value of member in <code>_codec_record_channel</code> .
<i>rightRecord-Channel</i>	audio codec record channel, reference <code>_codec_record_channel</code> , can be a value or combine of member in <code>_codec_record_channel</code> .

Returns

`kStatus_Success` is success, else configure failed.

75.4.6.2.19 `static status_t HAL_CODEC_SetPlay (void * handle, uint32_t playSource)`
[inline], [static]

Parameters

<i>handle</i>	codec handle.
<i>playSource</i>	audio codec play source, can be a value or combine value of <code>_codec_play_source</code> .

Returns

`kStatus_Success` is success, else configure failed.

75.4.6.2.20 `static status_t HAL_CODEC_ModuleControl (void * handle, uint32_t cmd, uint32_t data)`
[inline], [static]

This function is used for codec module control, support switch digital interface cmd, can be expand to support codec module specific feature

Parameters

<i>handle</i>	codec handle.
<i>cmd</i>	module control cmd, reference <code>_codec_module_ctrl_cmd</code> .
<i>data</i>	value to write, when cmd is <code>kCODEC_ModuleRecordSourceChannel</code> , the data should be a value combine of channel and source, please reference macro <code>CODEC_MODULE_RECORD_SOURCE_CHANNEL(source, LP, LN, RP, RN)</code> , reference codec specific driver for detail configurations.

Returns

`kStatus_Success` is success, else configure failed.

Chapter 76

Serial Manager

76.1 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](#) (0U)
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_MANAGER_TASK_HANDLE_TX](#) (0U)
Enable or disable SerialManager_Task() handle TX to prevent recursive calling.

- #define `SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE` (1U)
Set the default delay time in ms used by `SerialManager_WriteTimeDelay()`.
- #define `SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE` (1U)
Set the default delay time in ms used by `SerialManager_ReadTimeDelay()`.
- #define `SERIAL_MANAGER_TASK_HANDLE_RX_AVAILABLE_NOTIFY` (0U)
Enable or disable `SerialManager_Task()` handle RX data available notify.
- #define `SERIAL_MANAGER_WRITE_HANDLE_SIZE` (4U)
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 + 12U)
Macro to determine whether use common task.
- #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)
callback function

Enumerations

- enum `serial_port_type_t` {
`kSerialPort_Uart` = 1U,
`kSerialPort_UsbCdc`,
`kSerialPort_Swo`,
`kSerialPort_Virtual`,
`kSerialPort_Rpmsg`,
`kSerialPort_UartDma`,
`kSerialPort_SpiMaster`,
`kSerialPort_SpiSlave`,
`kSerialPort_None` }
serial port type
- enum `serial_manager_type_t` {
`kSerialManager_NonBlocking` = 0x0U,
`kSerialManager_Blocking` = 0x8F41U }
serial manager type
- enum `serial_manager_status_t` {
`kStatus_SerialManager_Success` = `kStatus_Success`,
`kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,
`kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,
`kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,
`kStatus_SerialManager_Canceled`,
`kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,
`kStatus_SerialManager_RingBufferOverflow`,
`kStatus_SerialManager_NotConnected` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 7)` }
serial manager error code

Functions

- `serial_manager_status_t SerialManager_Init (serial_handle_t serialHandle, const serial_manager_config_t *config)`
Initializes a serial manager module with the serial manager handle and the user configuration structure.
- `serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)`
De-initializes the serial manager module instance.
- `serial_manager_status_t SerialManager_OpenWriteHandle (serial_handle_t serialHandle, serial_write_handle_t writeHandle)`
Opens a writing handle for the serial manager module.
- `serial_manager_status_t SerialManager_CloseWriteHandle (serial_write_handle_t writeHandle)`
Closes a writing handle for the serial manager module.
- `serial_manager_status_t SerialManager_OpenReadHandle (serial_handle_t serialHandle, serial_read_handle_t readHandle)`
Opens a reading handle for the serial manager module.
- `serial_manager_status_t SerialManager_CloseReadHandle (serial_read_handle_t readHandle)`
Closes a reading for the serial manager module.

- `serial_manager_status_t` `SerialManager_WriteBlocking` (`serial_write_handle_t` writeHandle, `uint8_t` *buffer, `uint32_t` length)
Transmits data with the blocking mode.
- `serial_manager_status_t` `SerialManager_ReadBlocking` (`serial_read_handle_t` readHandle, `uint8_t` *buffer, `uint32_t` length)
Reads data with the blocking mode.
- `serial_manager_status_t` `SerialManager_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.
- static bool `SerialManager_needPollingIsr` (void)
Check if need polling ISR.

76.2 Data Structure Documentation

76.2.1 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.

76.2.2 struct serial_manager_callback_message_t

Data Fields

- `uint8_t` * `buffer`
Transferred buffer.
- `uint32_t` `length`
Transferred data length.

76.3 Macro Definition Documentation

76.3.1 #define SERIAL_MANAGER_WRITE_TIME_DELAY_DEFAULT_VALUE (1U)

76.3.2 #define SERIAL_MANAGER_READ_TIME_DELAY_DEFAULT_VALUE (1U)

76.3.3 #define SERIAL_MANAGER_USE_COMMON_TASK (0U)

Macro to determine whether use common task.

76.3.4 #define SERIAL_MANAGER_HANDLE_SIZE (SERIAL_MANAGER_HANDLE_SIZE_TEMP + 12U)

Definition of serial manager handle size.

**76.3.5 #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.
-------------	---

**76.3.6 #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.
-------------	---

76.3.7 #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.
-------------	--

76.3.8 #define SERIAL_MANAGER_TASK_PRIORITY (2U)

76.3.9 #define SERIAL_MANAGER_TASK_STACK_SIZE (1000U)

76.4 Enumeration Type Documentation

76.4.1 enum serial_port_type_t

Enumerator

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_None Serial port is none.

76.4.2 enum serial_manager_type_t

Enumerator

kSerialManager_NonBlocking None blocking handle.

kSerialManager_Blocking Blocking handle.

76.4.3 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.

76.5 Function Documentation

76.5.1 serial_manager_status_t SerialManager_Init (serial_handle_t *serialHandle*, const serial_manager_config_t * *config*)

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter *serialHandle* is a pointer to point to a memory space of size [SERIAL_MANAGER_HANDLE_SIZE](#) allocated by the caller. The Serial Manager module supports three types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc), USB CDC and swo. Please refer to [serial_port_type_t](#) for serial port setting. These three types can be set by using [serial_manager_config_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
```

```

*  uartConfig.clockRate = 24000000;
*  uartConfig.baudRate = 115200;
*  uartConfig.parityMode = kSerialManager_UartParityDisabled;
*  uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
*  uartConfig.enableRx = 1;
*  uartConfig.enableTx = 1;
*  uartConfig.enableRxRTS = 0;
*  uartConfig.enableTxCTS = 0;
*  config.portConfig = &uartConfig;
*  SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

For USB CDC,

```

*  #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
*  static SERIAL_MANAGER_HANDLE_DEFINE(s_serialHandle);
*  static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
*  serial_manager_config_t config;
*  serial_port_usb_cdc_config_t usbCdcConfig;
*  config.type = kSerialPort_UsbCdc;
*  config.ringBuffer = &s_ringBuffer[0];
*  config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
*  usbCdcConfig.controllerIndex =
*      kSerialManager_UsbControllerKhci0;
*  config.portConfig = &usbCdcConfig;
*  SerialManager_Init((serial_handle_t)s_serialHandle, &config);
*

```

Parameters

<i>serialHandle</i>	Pointer to point to a memory space of size SERIAL_MANAGER_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: SERIAL_MANAGER_HANDLE_DEFINE(serialHandle) ; or <code>uint32_t serialHandle[((SERIAL_MANAGER_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t))];</code>
<i>config</i>	Pointer to user-defined configuration structure.

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The Serial Manager module initialization succeed.

76.5.2 serial_manager_status_t SerialManager_Deinit (serial_handle_t serialHandle)

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return `kStatus_SerialManager_Busy`.

Parameters

<i>serialHandle</i>	The serial manager module handle pointer.
---------------------	---

Return values

<i>kStatus_SerialManager_Success</i>	The serial manager de-initialization succeed.
<i>kStatus_SerialManager_Busy</i>	Opened reading or writing handle is not closed.

76.5.3 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);
*
```

76.5.4 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.
--------------------------------------	-------------------------------

76.5.5 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 <code>uint32_t readHandle[(((SERIAL_MANAGER_READ_HANDLE_SIZE + sizeof(uint32_t) - 1U) / sizeof(uint32_t)))]</code> ;

Return values

<i>kStatus_SerialManager_Error</i>	An error occurred.
<i>kStatus_SerialManager_Success</i>	The reading handle is opened.
<i>kStatus_SerialManager_Busy</i>	Previous reading handle is not closed.

Example below shows how to use this API to read data.

```
*  static SERIAL_MANAGER_READ_HANDLE_DEFINE(s_serialReadHandle);
*  SerialManager_OpenReadHandle((serial_handle_t)serialHandle,
*    (serial_read_handle_t)s_serialReadHandle);
*  static uint8_t s_nonBlockingBuffer[64];
*  SerialManager_InstallRxCallback((serial_read_handle_t)s_serialReadHandle,
*    APP_SerialManagerRxCallback,
*    s_serialReadHandle);
*  SerialManager_ReadNonBlocking((serial_read_handle_t)s_serialReadHandle,
*    s_nonBlockingBuffer,
*    sizeof(s_nonBlockingBuffer));
*
```

76.5.6 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.
--------------------------------------	-------------------------------

76.5.7 serial_manager_status_t SerialManager_WriteBlocking (serial_manager_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.

76.5.8 serial_manager_status_t SerialManager_ReadBlocking (serial_manager_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_</i> <i>Success</i>	Successfully received all data.
<i>kStatus_SerialManager_</i> <i>Busy</i>	Previous transmission still not finished; data not all received yet.
<i>kStatus_SerialManager_</i> <i>Error</i>	An error occurred.

76.5.9 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_</i> <i>Success</i>	Successful operation.
---	-----------------------

76.5.10 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.
--------------------------------------	-----------------------

76.5.11 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.
-------------	------------------

76.6 Serial Port Uart

76.6.1 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

76.6.2 Enumeration Type Documentation

76.6.2.1 enum serial_port_uart_parity_mode_t

Enumerator

kSerialManager_UartParityDisabled Parity disabled.
kSerialManager_UartParityEven Parity even enabled.
kSerialManager_UartParityOdd Parity odd enabled.

76.6.2.2 enum serial_port_uart_stop_bit_count_t

Enumerator

kSerialManager_UartOneStopBit One stop bit.
kSerialManager_UartTwoStopBit Two stop bits.

76.7 Serial Port USB

76.7.1 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.

76.7.2 Data Structure Documentation

76.7.2.1 struct serial_port_usb_cdc_config_t

Data Fields

- [serial_port_usb_cdc_controller_index_t](#) controllerIndex
controller index

76.7.3 Enumeration Type Documentation

76.7.3.1 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.

76.7.4 USB Device Configuration

76.8 Serial Port SWO

76.8.1 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

76.8.2 Data Structure Documentation

76.8.2.1 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.

76.8.3 Enumeration Type Documentation

76.8.3.1 enum serial_port_swo_protocol_t

Enumerator

kSerialManager_SwoProtocolManchester SWO Manchester protocol.
kSerialManager_SwoProtocolNrz SWO UART/NRZ protocol.

Chapter 77

Gpc

77.1 Overview

Data Structures

- struct `gpc_tran_step_config_t`
Configuration for GPC transition step. [More...](#)

Enumerations

- enum {
 `kGPC_CM_EventWakeupRequest`,
 `kGPC_CM_DebugWakeupRequest` }
 _gpc_cm_non_irq_wakeup_request GPC Non-IRQ wakeup request.
- enum {
 `kGPC_SetPoint0` = 1UL << 0UL,
 `kGPC_SetPoint1` = 1UL << 1UL,
 `kGPC_SetPoint2` = 1UL << 2UL,
 `kGPC_SetPoint3` = 1UL << 3UL,
 `kGPC_SetPoint4` = 1UL << 4UL,
 `kGPC_SetPoint5` = 1UL << 5UL,
 `kGPC_SetPoint6` = 1UL << 6UL,
 `kGPC_SetPoint7` = 1UL << 7UL,
 `kGPC_SetPoint8` = 1UL << 8UL,
 `kGPC_SetPoint9` = 1UL << 9UL,
 `kGPC_SetPoint10` = 1UL << 10UL,
 `kGPC_SetPoint11` = 1UL << 11UL,
 `kGPC_SetPoint12` = 1UL << 12UL,
 `kGPC_SetPoint13` = 1UL << 13UL,
 `kGPC_SetPoint14` = 1UL << 14UL,
 `kGPC_SetPoint15` = 1UL << 15UL }
• enum
 _gpc_cm_interrupt_status_flag
- enum `gpc_cm_standby_mode_status_t` {
 `kGPC_CM_SleepBusy`,
 `kGPC_CM_WakeupBusy` }
 CPU standby mode status.
- 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_SleepSP = 6UL,
kGPC_CM_SleepSTBY = 7UL,
kGPC_CM_WakeupSTBY = 8UL,
kGPC_CM_WakeupSP = 9UL,
kGPC_CM_WakeupPower = 10UL,
kGPC_CM_WakeupReset = 11UL,
kGPC_CM_WakeupIso = 12UL,
kGPC_CM_WakeupPll = 13UL,
kGPC_CM_WakeupLpcg = 14UL,
kGPC_CM_WakeupSsar = 15UL }

```
- CPU mode transition step in sleep/wakeup sequence.*
- enum `gpc_tran_step_counter_mode_t` {

```

kGPC_StepCounterDisableMode,
kGPC_StepCounterDelayMode,
kGPC_StepCounterIgnoreResponseMode = 2UL,
kGPC_StepCounterTimeOutMode = 3UL }

```
- Step counter work mode.*
- enum `gpc_sp_tran_step_t` {

```

kGPC_SP_SsarSave = 0UL,
kGPC_SP_LpcgOff = 1UL,
kGPC_SP_GroupDown = 2UL,
kGPC_SP_RootDown = 3UL,
kGPC_SP_PllOff = 4UL,
kGPC_SP_IsoOn = 5UL,
kGPC_SP_ResetEarly = 6UL,
kGPC_SP_PowerOff = 7UL,
kGPC_SP_BiasOff = 8UL,
kGPC_SP_BandgapPllLdoOff = 9UL,
kGPC_SP_LdoPre = 10UL,
kGPC_SP_DcdcDown = 11UL,
kGPC_SP_DcdcUp = 12UL,
kGPC_SP_LdoPost = 13UL,
kGPC_SP_BandgapPllLdoOn = 14UL,
kGPC_SP_BiasOn = 15UL,
kGPC_SP_PowerOn = 16UL,
kGPC_SP_ResetLate = 17UL,
kGPC_SP_IsoOff = 18UL,
kGPC_SP_PllOn = 19UL,
kGPC_SP_RootUp = 20UL,
kGPC_SP_GroupUp = 21UL,
kGPC_SP_LpcgOn = 22UL,
kGPC_SP_SsarRestore = 23UL }

```

*GPC set point transition steps.*

- enum `gpc_cpu_mode_t` {  
`kGPC_RunMode` = 0x0UL,  
`kGPC_WaitMode` = 0x1UL,  
`kGPC_StopMode` = 0x2UL,  
`kGPC_SuspendMode` = 0x3UL }

*CPU mode.*

- enum `gpc_cm_wakeup_sp_sel_t` {  
`kGPC_CM_WakeupSetpoint`,  
`kGPC_CM_RequestPreviousSetpoint` = 1UL }

*CPU wakeup sequence setpoint options.*

- enum `gpc_stby_tran_step_t` {

```

kGPC_STBY_LpcgIn = 0UL,
kGPC_STBY_PllIn = 1UL,
kGPC_STBY_BiasIn = 2UL,
kGPC_STBY_PldoIn = 3UL,
kGPC_STBY_BandgapIn = 4UL,
kGPC_STBY_LdoIn = 5UL,
kGPC_STBY_DcdcIn = 6UL,
kGPC_STBY_PmicIn = 7UL,
kGPC_STBY_PmicOut = 8UL,
kGPC_STBY_DcdcOut = 9UL,
kGPC_STBY_LdoOut = 10UL,
kGPC_STBY_BandgapOut = 11UL,
kGPC_STBY_PldoOut = 12UL,
kGPC_STBY_BiasOut = 13UL,
kGPC_STBY_PllOut = 14UL,
kGPC_STBY_LpcgOut = 15UL }

```

*GPC standby mode transition steps.*

## Driver version

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

## CPU mode control

- static void **GPC\_CM\_EnableCpuSleepHold** (GPC\_CPU\_MODE\_CTRL\_Type \*base, bool enable)
- static void **GPC\_CM\_SetNextCpuMode** (GPC\_CPU\_MODE\_CTRL\_Type \*base, **gpc\_cpu\_mode\_t** mode)  
*Set the CPU mode on the next sleep event.*
- static **gpc\_cpu\_mode\_t** **GPC\_CM\_GetCurrentCpuMode** (GPC\_CPU\_MODE\_CTRL\_Type \*base)  
*Get current CPU mode.*
- static **gpc\_cpu\_mode\_t** **GPC\_CM\_GetPreviousCpuMode** (GPC\_CPU\_MODE\_CTRL\_Type \*base)  
*Get previous CPU mode.*
- void **GPC\_CM\_EnableIrqWakeup** (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t irqId, bool enable)  
*Enable IRQ wakeup request.*
- static void **GPC\_CM\_EnableNonIrqWakeup** (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t mask, bool enable)  
*Enable Non-IRQ wakeup request.*
- bool **GPC\_CM\_GetIrqWakeupStatus** (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t irqId)  
*Get the status of the IRQ wakeup request.*
- static bool **GPC\_CM\_GetNonIrqWakeupStatus** (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t mask)  
*Get the status of the Non-IRQ wakeup request.*
- void **GPC\_CM\_ConfigCpuModeTransitionStep** (GPC\_CPU\_MODE\_CTRL\_Type \*base, **gpc\_cm\_tran\_step\_t** step, const **gpc\_tran\_step\_config\_t** \*config)  
*Config the cpu mode transition step.*



- void [GPC\\_CM\\_RequestSleepModeSetPointTransition](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint8\_t setPointSleep, uint8\_t setPointWakeup, [gpc\\_cm\\_wakeup\\_sp\\_sel\\_t](#) wakeupSel)  
*Request a set point transition before the CPU transfers into a sleep mode.*
- void [GPC\\_CM\\_RequestRunModeSetPointTransition](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint8\_t setPointRun)  
*Request a set point transition during run mode.*
- static void [GPC\\_CM\\_SetSetPointMapping](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t setPoint, uint32\_t map)  
*Set the set point mapping value for each set point.*
- void [GPC\\_CM\\_SetCpuModeSetPointMapping](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, [gpc\\_cpu\\_mode\\_t](#) mode, uint32\_t map)  
*Set the set point mapping value for each cpu mode.*
- void [GPC\\_CM\\_RequestStandbyMode](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, const [gpc\\_cpu\\_mode\\_t](#) mode)  
*Request the chip into standby mode.*
- void [GPC\\_CM\\_ClearStandbyModeRequest](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, const [gpc\\_cpu\\_mode\\_t](#) mode)  
*Clear the standby mode request.*
- static bool [GPC\\_CM\\_GetStandbyModeStatus](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t mask)  
*Get the status of the CPU standby mode transition.*
- static uint32\_t [GPC\\_CM\\_GetInterruptStatusFlags](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base)  
*Get the status flags of the GPC CPU module.*
- void [GPC\\_CM\\_ClearInterruptStatusFlags](#) (GPC\_CPU\_MODE\_CTRL\_Type \*base, uint32\_t mask)  
*Clears CPU module interrupt status flags.*

## Set point request control

- static void [GPC\\_SP\\_SetSetpointPriority](#) (GPC\_SET\_POINT\_CTRL\_Type \*base, uint32\_t setPoint, uint32\_t priority)  
*Set the priority of set point.*
- void [GPC\\_SP\\_ConfigSetPointTransitionStep](#) (GPC\_SET\_POINT\_CTRL\_Type \*base, [gpc\\_sp\\_tran\\_step\\_t](#) step, const [gpc\\_tran\\_step\\_config\\_t](#) \*config)  
*Config the set point transition step.*
- static uint8\_t [GPC\\_SP\\_GetCurrentSetPoint](#) (GPC\_SET\_POINT\_CTRL\_Type \*base)  
*Get system current setpoint, only valid when setpoint trans not busy.*
- static uint8\_t [GPC\\_SP\\_GetPreviousSetPoint](#) (GPC\_SET\_POINT\_CTRL\_Type \*base)  
*Get system previous setpoint, only valid when setpoint trans not busy.*
- static uint8\_t [GPC\\_SP\\_GetTargetSetPoint](#) (GPC\_SET\_POINT\_CTRL\_Type \*base)  
*Get target setpoint.*

## Standby mode control

- void [GPC\\_STBY\\_ConfigStandbyTransitionStep](#) (GPC\_STBY\_CTRL\_Type \*base, [gpc\\_stby\\_tran\\_step\\_t](#) step, const [gpc\\_tran\\_step\\_config\\_t](#) \*config)  
*Config the standby transition step.*

## 77.2 Data Structure Documentation

## 77.2.1 struct gpc\_tran\_step\_config\_t

### Data Fields

- uint32\_t [stepCount](#)  
*Step count, which is depended on the value of cntMode.*
- [gpc\\_tran\\_step\\_counter\\_mode\\_t](#) cntMode  
*Step counter working mode.*
- bool [enableStep](#)  
*Enable the step.*

### Field Documentation

- (1) uint32\_t gpc\_tran\_step\_config\_t::stepCount
- (2) gpc\_tran\_step\_counter\_mode\_t gpc\_tran\_step\_config\_t::cntMode
- (3) bool gpc\_tran\_step\_config\_t::enableStep

## 77.3 Macro Definition Documentation

### 77.3.1 #define FSL\_GPC\_RIVER\_VERSION (MAKE\_VERSION(2, 2, 0))

## 77.4 Enumeration Type Documentation

### 77.4.1 anonymous enum

#### Enumerator

- kGPC\_CM\_EventWakeupRequest** Event wakeup request.  
**kGPC\_CM\_DebugWakeupRequest** Debug wakeup request.

### 77.4.2 anonymous enum

#### Enumerator

- kGPC\_SetPoint0** GPC set point 0.  
**kGPC\_SetPoint1** GPC set point 1.  
**kGPC\_SetPoint2** GPC set point 2.  
**kGPC\_SetPoint3** GPC set point 3.  
**kGPC\_SetPoint4** GPC set point 4.  
**kGPC\_SetPoint5** GPC set point 5.  
**kGPC\_SetPoint6** GPC set point 6.  
**kGPC\_SetPoint7** GPC set point 7.  
**kGPC\_SetPoint8** GPC set point 8.  
**kGPC\_SetPoint9** GPC set point 9.  
**kGPC\_SetPoint10** GPC set point 10.

***kGPC\_SetPoint11*** GPC set point 11.  
***kGPC\_SetPoint12*** GPC set point 12.  
***kGPC\_SetPoint13*** GPC set point 13.  
***kGPC\_SetPoint14*** GPC set point 14.  
***kGPC\_SetPoint15*** GPC set point 15.

### 77.4.3 enum gpc\_cm\_standby\_mode\_status\_t

Enumerator

***kGPC\_CM\_SleepBusy*** Indicate the CPU is busy entering standby mode.  
***kGPC\_CM\_WakeupBusy*** Indicate the CPU is busy exiting standby mode.

### 77.4.4 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\_SleepSP*** Setpoint sleep step. Note that this step is controlled by setpoint controller.  
***kGPC\_CM\_SleepSTBY*** Standby sleep step. Note that this step is controlled by standby controller.  
***kGPC\_CM\_WakeupSTBY*** Standby wakeup step. Note that this step is controlled by standby controller.  
***kGPC\_CM\_WakeupSP*** Setpoint wakeup step. Note that this step is controlled by setpoint 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.

### 77.4.5 enum gpc\_tran\_step\_counter\_mode\_t

Enumerator

***kGPC\_StepCounterDisableMode*** Counter disable mode: not use step counter, step completes once receiving step\_done.

***kGPC\_StepCounterDelayMode*** Counter delay mode: delay after receiving step\_done, delay cycle number is STEP\_CNT.

***kGPC\_StepCounterIgnoreResponseMode*** Ignore step\_done response, the counter starts to count once step begins, when counter reaches STEP\_CNT value, the step completes.

***kGPC\_StepCounterTimeOutMode*** Time out mode, the counter starts to count once step begins, the step completes when either step\_done received or counting to STEP\_CNT value.

## 77.4.6 enum gpc\_sp\_tran\_step\_t

Enumerator

***kGPC\_SP\_SsarSave*** SSAR save step.

***kGPC\_SP\_LpcgOff*** LPCG off step.

***kGPC\_SP\_GroupDown*** Group down step.

***kGPC\_SP\_RootDown*** Root down step.

***kGPC\_SP\_PllOff*** PLL off step.

***kGPC\_SP\_IsoOn*** ISO on.

***kGPC\_SP\_ResetEarly*** Reset early step.

***kGPC\_SP\_PowerOff*** Power off step.

***kGPC\_SP\_BiasOff*** Bias off step.

***kGPC\_SP\_BandgapPllLdoOff*** Bandgap and PLL\_LDO off step.

***kGPC\_SP\_LdoPre*** LDO (Low-Dropout) pre step.

***kGPC\_SP\_DcdcDown*** DCDC down step.

***kGPC\_SP\_DcdcUp*** DCDC up step.

***kGPC\_SP\_LdoPost*** LDO post step.

***kGPC\_SP\_BandgapPllLdoOn*** Bandgap and PLL\_LDO on step.

***kGPC\_SP\_BiasOn*** Bias on step.

***kGPC\_SP\_PowerOn*** Power on step.

***kGPC\_SP\_ResetLate*** Reset late step.

***kGPC\_SP\_IsoOff*** ISO off step.

***kGPC\_SP\_PllOn*** PLL on step.

***kGPC\_SP\_RootUp*** Root up step.

***kGPC\_SP\_GroupUp*** Group up step.

***kGPC\_SP\_LpcgOn*** LPCG on step.

***kGPC\_SP\_SsarRestore*** SSAR restore step.

## 77.4.7 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.

#### 77.4.8 enum gpc\_cm\_wakeup\_sp\_sel\_t

Enumerator

***kGPC\_CM\_WakeupSetpoint*** Request SP transition to CPU\_SP\_WAKEUP (param "setPoint-Wakeup" in gpc\_cm\_sleep\_sp\_tran\_config\_t).

***kGPC\_CM\_RequestPreviousSetpoint*** Request SP transition to the set point when the sleep event happens.

#### 77.4.9 enum gpc\_stby\_tran\_step\_t

Enumerator

***kGPC\_STBY\_LpcgIn*** LPCG in step.

***kGPC\_STBY\_PllIn*** PLL in step.

***kGPC\_STBY\_BiasIn*** Bias in step.

***kGPC\_STBY\_PldoIn*** PLDO in step.

***kGPC\_STBY\_BandgapIn*** Bandgap in step.

***kGPC\_STBY\_LdoIn*** LDO in step.

***kGPC\_STBY\_DcdcIn*** DCDC in step.

***kGPC\_STBY\_PmicIn*** PMIC in step.

***kGPC\_STBY\_PmicOut*** PMIC out step.

***kGPC\_STBY\_DcdcOut*** DCDC out step.

***kGPC\_STBY\_LdoOut*** LDO out step.

***kGPC\_STBY\_BandgapOut*** Bandgap out step.

***kGPC\_STBY\_PldoOut*** PLDO out step.

***kGPC\_STBY\_BiasOut*** Bias out step.

***kGPC\_STBY\_PllOut*** PLL out step.

***kGPC\_STBY\_LpcgOut*** LPCG out step.

### 77.5 Function Documentation

**77.5.1 static void GPC\_CM\_SetNextCpuMode ( GPC\_CPU\_MODE\_CTRL\_Type \*  
base, 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>base</i> | GPC CPU module base address.                                            |
| <i>mode</i> | The CPU mode that the core will transmit to, refer to "gpc_cpu_mode_t". |

### 77.5.2 static gpc\_cpu\_mode\_t GPC\_CM\_GetCurrentCpuMode ( GPC\_CPU\_MODE\_CTRL\_Type \* *base* ) [inline], [static]

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | GPC CPU module base address. |
|-------------|------------------------------|

## Returns

The current CPU mode, in type of [gpc\\_cpu\\_mode\\_t](#).

### 77.5.3 static gpc\_cpu\_mode\_t GPC\_CM\_GetPreviousCpuMode ( GPC\_CPU\_MODE\_CTRL\_Type \* *base* ) [inline], [static]

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | GPC CPU module base address. |
|-------------|------------------------------|

## Returns

The previous CPU mode, in type of [gpc\\_cpu\\_mode\\_t](#).

### 77.5.4 void GPC\_CM\_EnableIrqWakeup ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *irqId*, bool *enable* )

This function enables the IRQ request which can wakeup the CPU platform.

## Parameters

---

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | GPC CPU module base address.              |
| <i>irqId</i>  | ID of the IRQ, accessible range is 0-255. |
| <i>enable</i> | Enable the IRQ request or not.            |

**77.5.5 static void GPC\_CM\_EnableNonIrqWakeup ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *mask*, bool *enable* ) [inline], [static]**

This function enables the non-IRQ request which can wakeup the CPU platform.

Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | GPC CPU module base address.                             |
| <i>mask</i>   | Non-IRQ type, refer to "_gpc_cm_non_irq_wakeup_request". |
| <i>enable</i> | Enable the Non-IRQ request or not.                       |

**77.5.6 bool GPC\_CM\_GetIrqWakeupStatus ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *irqId* )**

Parameters

|              |                                           |
|--------------|-------------------------------------------|
| <i>base</i>  | GPC CPU module base address.              |
| <i>irqId</i> | ID of the IRQ, accessible range is 0-255. |

Returns

Indicate the IRQ request is asserted or not.

**77.5.7 static bool GPC\_CM\_GetNonIrqWakeupStatus ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                          |
|-------------|----------------------------------------------------------|
| <i>base</i> | GPC CPU module base address.                             |
| <i>mask</i> | Non-IRQ type, refer to "_gpc_cm_non_irq_wakeup_request". |

## Returns

Indicate the Non-IRQ request is asserted or not.

**77.5.8 void GPC\_CM\_ConfigCpuModeTransitionStep ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, gpc\_cm\_tran\_step\_t *step*, const gpc\_tran\_step\_config\_t \* *config* )**

## Note

This function can not config the setpoint sleep/wakeup operation for those operation is controlled by setpoint control. This function can not config the standby sleep/wakeup too, because those operation is controlled by standby controlled.

## Parameters

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| <i>base</i>   | GPC CPU module base address.                                      |
| <i>step</i>   | step type, refer to "gpc_cm_tran_step_t".                         |
| <i>config</i> | transition step configuration, refer to "gpc_tran_step_config_t". |

**77.5.9 void GPC\_CM\_RequestSleepModeSetPointTransition ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint8\_t *setPointSleep*, uint8\_t *setPointWakeup*, gpc\_cm\_wakeup\_sp\_sel\_t *wakeupSel* )**

This function triggers the set point transition during a CPU Sleep/wakeup event and selects which one the CMC want to transfer to.

## Parameters

|                      |                                                                                      |
|----------------------|--------------------------------------------------------------------------------------|
| <i>base</i>          | GPC CPU module base address.                                                         |
| <i>setPointSleep</i> | The set point CPU want the system to transit to on next CPU platform sleep sequence. |



|                       |                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------|
| <i>setPointWakeup</i> | The set point CPU want the system to transit to on next CPU platform wakeup sequence. |
| <i>wakeupSel</i>      | Select the set point transition on the next CPU platform wakeup sequence.             |

#### 77.5.10 void GPC\_CM\_RequestRunModeSetPointTransition ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint8\_t *setPointRun* )

This function triggers the set point transition and selects which one the CMC want to transfer to.

Parameters

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>base</i>        | GPC CPU module base address.                                  |
| <i>setPointRun</i> | The set point CPU want the system to transit in the run mode. |

#### 77.5.11 static void GPC\_CM\_SetSetPointMapping ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *setPoint*, uint32\_t *map* ) [inline], [static]

This function configures which set point is allowed after current set point. If there are multiple setpoints, use:

```
* map = kGPC_SetPoint0 | kGPC_SetPoint1 | ... |
 kGPC_SetPoint15;
*
```

Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>base</i>     | GPC CPU module base address.                              |
| <i>setPoint</i> | Set point index, available range is 0-15.                 |
| <i>map</i>      | Map value of the set point. Refer to "_gpc_setpoint_map". |

#### 77.5.12 void GPC\_CM\_SetCpuModeSetPointMapping ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, gpc\_cpu\_mode\_t *mode*, uint32\_t *map* )

This function configures which set point is allowed when CPU enters RUN/WAIT/STOP/SUSPEND. If there are multiple setpoints, use:

```
* map = kkGPC_SetPoint0 | kGPC_SetPoint1 | ... |
 kGPC_SetPoint15;
*
```

## Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| <i>base</i> | GPC CPU module base address.                              |
| <i>mode</i> | CPU mode. Refer to "gpc_cpu_mode_t".                      |
| <i>map</i>  | Map value of the set point. Refer to "_gpc_setpoint_map". |

**77.5.13 void GPC\_CM\_RequestStandbyMode ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, const gpc\_cpu\_mode\_t *mode* )**

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | GPC CPU module base address.         |
| <i>mode</i> | CPU mode. Refer to "gpc_cpu_mode_t". |

**77.5.14 void GPC\_CM\_ClearStandbyModeRequest ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, const gpc\_cpu\_mode\_t *mode* )**

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | GPC CPU module base address.         |
| <i>mode</i> | CPU mode. Refer to "gpc_cpu_mode_t". |

**77.5.15 static bool GPC\_CM\_GetStandbyModeStatus ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Parameters

|             |                                                                               |
|-------------|-------------------------------------------------------------------------------|
| <i>base</i> | GPC CPU module base address.                                                  |
| <i>mask</i> | Standby mode transition status mask, refer to "gpc_cm_standby_mode_status_t". |

## Returns

Indicate the CPU's standby transition status.

**77.5.16** `static uint32_t GPC_CM_GetInterruptStatusFlags ( GPC_CPU_MODE_CTRL_Type * base ) [inline], [static]`

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | GPC CPU module base address. |
|-------------|------------------------------|

## Returns

The OR'ed value of status flags.

**77.5.17 void GPC\_CM\_ClearInterruptStatusFlags ( GPC\_CPU\_MODE\_CTRL\_Type \* *base*, uint32\_t *mask* )**

## Parameters

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| <i>base</i> | GPC CPU module base address.                                                                          |
| <i>mask</i> | The interrupt status flags to be cleared. Should be the OR'ed value of _gpc_cm_interrupt_status_flag. |

**77.5.18 static void GPC\_SP\_SetSetpointPriority ( GPC\_SET\_POINT\_CTRL\_Type \* *base*, uint32\_t *setPoint*, uint32\_t *priority* ) [inline], [static]**

This function will configure the priority of the set point. If the result of API GPC\_SP\_GetAllowedSetPointMap() has more than one valid bit, high priority set point will be taken.

## Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>base</i>     | GPC Setpoint controller base address.     |
| <i>setPoint</i> | Set point index, available range is 0-15. |
| <i>priority</i> | Priority level, available range is 0-15.  |

**77.5.19 void GPC\_SP\_ConfigSetPointTransitionStep ( GPC\_SET\_POINT\_CTRL\_Type \* *base*, gpc\_sp\_tran\_step\_t *step*, const gpc\_tran\_step\_config\_t \* *config* )**

## Parameters

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| <i>base</i>   | GPC Setpoint controller base address.                             |
| <i>step</i>   | step type, refer to "gpc_sp_tran_step_t".                         |
| <i>config</i> | transition step configuration, refer to "gpc_tran_step_config_t". |

**77.5.20 static uint8\_t GPC\_SP\_GetCurrentSetPoint ( GPC\_SET\_POINT\_CTRL\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | GPC Setpoint controller base address. |
|-------------|---------------------------------------|

## Returns

The current setpoint number, range from 0 to 15.

**77.5.21 static uint8\_t GPC\_SP\_GetPreviousSetPoint ( GPC\_SET\_POINT\_CTRL\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | GPC Setpoint controller base address. |
|-------------|---------------------------------------|

## Returns

The previous setpoint number, range from 0 to 15.

**77.5.22 static uint8\_t GPC\_SP\_GetTargetSetPoint ( GPC\_SET\_POINT\_CTRL\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | GPC Setpoint controller base address. |
|-------------|---------------------------------------|

## Returns

The target setpoint number, range from 0 to 15.

**77.5.23 void GPC\_STBY\_ConfigStandbyTransitionStep ( GPC\_STBY\_CTRL\_Type \* *base*, gpc\_stby\_tran\_step\_t *step*, const gpc\_tran\_step\_config\_t \* *config* )**

## Parameters

|               |                                                                   |
|---------------|-------------------------------------------------------------------|
| <i>base</i>   | GPC Setpoint controller base address.                             |
| <i>step</i>   | step type, refer to "gpc_stby_tran_step_t".                       |
| <i>config</i> | transition step configuration, refer to "gpc_tran_step_config_t". |

# Chapter 78

## PGMC

### 78.1 Overview

#### Data Structures

- struct [pgmc\\_bpc\\_cpu\\_power\\_mode\\_option\\_t](#)  
*The control option of the power domain controlled by CPU power mode. [More...](#)*
- struct [pgmc\\_bpc\\_setpoint\\_mode\\_option\\_t](#)  
*The control option of the power domain controlled by setpoint mode. [More...](#)*

#### Enumerations

- enum {  
    [kPGMC\\_SetPoint0](#) = 1UL << 0UL,  
    [kPGMC\\_SetPoint1](#) = 1UL << 1UL,  
    [kPGMC\\_SetPoint2](#) = 1UL << 2UL,  
    [kPGMC\\_SetPoint3](#) = 1UL << 3UL,  
    [kPGMC\\_SetPoint4](#) = 1UL << 4UL,  
    [kPGMC\\_SetPoint5](#) = 1UL << 5UL,  
    [kPGMC\\_SetPoint6](#) = 1UL << 6UL,  
    [kPGMC\\_SetPoint7](#) = 1UL << 7UL,  
    [kPGMC\\_SetPoint8](#) = 1UL << 8UL,  
    [kPGMC\\_SetPoint9](#) = 1UL << 9UL,  
    [kPGMC\\_SetPoint10](#) = 1UL << 10UL,  
    [kPGMC\\_SetPoint11](#) = 1UL << 11UL,  
    [kPGMC\\_SetPoint12](#) = 1UL << 12UL,  
    [kPGMC\\_SetPoint13](#) = 1UL << 13UL,  
    [kPGMC\\_SetPoint14](#) = 1UL << 14UL,  
    [kPGMC\\_SetPoint15](#) = 1UL << 15UL }  
*The enumeration of setpoint.*
- enum [\\_pgmc\\_mif\\_signal\\_behaviour](#) {  
    [kPGMC\\_AssertSleepSignal](#) = 1U << 0U,  
    [kPGMC\\_AssertInputGateSignal](#) = 1U << 1U,  
    [kPGMC\\_AssertLowSpeedSignal](#) = 1U << 2U,  
    [kPGMC\\_AssertHighSpeedSignal](#) = 1U << 3U,  
    [kPGMC\\_AssertStandbySignal](#) = 1U << 4U,  
    [kPGMC\\_AssertArrayPowerDownSignal](#) = 1U << 5U,  
    [kPGMC\\_AssertPeripheralPowerDownSignal](#) = 1U << 6U,  
    [kPGMC\\_AssertInitnSignal](#) = 1U << 7U,  
    [kPGMC\\_AssertSwitch1OffSignal](#) = 1U << 8U,  
    [kPGMC\\_AssertSwitch2OffSignal](#) = 1U << 9U,  
    [kPGMC\\_AssertIsoSignal](#) = 1U << 10U }



*The enumeration of MIF signal behaviour(Such as Sleep Signal, Standby Signal, and so on).*

- enum `pgmc_bpc_assign_domain_t` {  
`kPGMC_CM7Core` = 0U,  
`kPGMC_CM4Core` = 1U }

*PGMC BPC assign domain enumeration.*

- enum `pgmc_cpu_mode_t` {  
`kPGMC_RunMode` = 0x0UL,  
`kPGMC_WaitMode` = 0x1UL,  
`kPGMC_StopMode` = 0x2UL,  
`kPGMC_SuspendMode` = 0x3UL }

*CPU mode.*

- enum `pgmc_control_mode_t`

*PGMC control modes.*

- enum `pgmc_memory_low_power_level_t` {  
`kPGMC_MLPLHighSpeed` = 1U,  
`kPGMC_MLPLNormal` = 3U,  
`kPGMC_MLPLLowSpeed` = 4U,  
`kPGMC_MLPLInputGating` = 5U,  
`kPGMC_MLPLStandby` = 6U,  
`kPGMC_MLPLSleep` = 8U,  
`kPGMC_MLPLArrOnPerOff` = 9U,  
`kPGMC_MLPLArrOffPerOn` = 10U,  
`kPGMC_MLPLArrOffPerOff` = 11U,  
`kPGMC_MLPLSw2` = 13U,  
`kPGMC_MLPLSw2PerOff` = 14U,  
`kPGMC_MLPLSw1PerOff` = 15U }

*The enumeration of memory low power level.*

- enum `pgmc_mif_signal_t` {  
`kPGMC_SleepSignal` = 0U,  
`kPGMC_InputGateSignal` = 1U,  
`kPGMC_LowSpeedSignal` = 2U,  
`kPGMC_HighSpeedSignal` = 3U,  
`kPGMC_StandbySignal` = 4U,  
`kPGMC_ArrayPowerDownSignal` = 5U,  
`kPGMC_PeripheralPowerDownSignal` = 6U,  
`kPGMC_InitnSignal` = 7U,  
`kPGMC_Switch1OffSignal` = 8U,  
`kPGMC_Switch2OffSignal` = 9U,  
`kPGMC_IsoSignal` = 10U }

*The enumeration of MIF signal.*

## Driver version

- #define `FSL_PGMC_RIVER_VERSION` (MAKE\_VERSION(2, 1, 1))  
*PGMC driver version 2.1.1.*

## Basic Power Controller Related Interfaces

- void [PGMC\\_BPC\\_ControlPowerDomainByCpuPowerMode](#) (PGMC\_BPC\_Type \*base, [pgmc\\_cpu\\_mode\\_t](#) mode, const [pgmc\\_bpc\\_cpu\\_power\\_mode\\_option\\_t](#) \*option)  
*Makes the BPC module controlled by the target CPU power mode, such as Wait mode.*
- void [PGMC\\_BPC\\_ControlPowerDomainBySetPointMode](#) (PGMC\_BPC\_Type \*base, uint32\_t set-PointMap, const [pgmc\\_bpc\\_setpoint\\_mode\\_option\\_t](#) \*option)  
*Makes the BPC module controlled by the target set points.*
- void [PGMC\\_BPC\\_ControlPowerDomainBySoftwareMode](#) (PGMC\_BPC\_Type \*base, bool power-Off)  
*Controls the selected power domain by software mode.*
- static void [PGMC\\_BPC\\_DisableLowPower](#) (PGMC\_BPC\_Type \*base)  
*Disables low power mode control.*
- static void [PGMC\\_BPC\\_RequestStateRestoreAtRunMode](#) (PGMC\_BPC\_Type \*base)  
*Requests power domain state restore at run mode.*
- static void [PGMC\\_BPC\\_RequestStateRestoreAtSetPoint](#) (PGMC\_BPC\_Type \*base, uint32\_t set-PointMap)  
*Requests power domain state restore when enters the selected set points.*
- static void [PGMC\\_BPC-AllowUserModeAccess](#) (PGMC\_BPC\_Type \*base, bool enable)  
*Allows user mode access or not for the BPC module.*
- static void [PGMC\\_BPC-AllowNonSecureModeAccess](#) (PGMC\_BPC\_Type \*base, bool enable)  
*Allows non-secure mode access or not for the BPC module.*
- static void [PGMC\\_BPC\\_LockAccessSetting](#) (PGMC\_BPC\_Type \*base)  
*Locks access related settings for the BPC module, including Secure access setting and user access setting.*
- static void [PGMC\\_BPC\\_SetDomainIdWhiteList](#) (PGMC\_BPC\_Type \*base, uint8\_t domainId)  
*Sets the corresponding domain ID that can access CPU mode control registers for the BPC module.*
- static void [PGMC\\_BPC\\_LockDomainIDWhiteList](#) (PGMC\_BPC\_Type \*base)  
*Locks the value of Domain ID white list for the BPC module.*
- static void [PGMC\\_BPC\\_LockLowPowerConfigurationFields](#) (PGMC\_BPC\_Type \*base)  
*Locks low power configuration fields for the BPC module.*

## CPU Power Controller Related Interfaces

- void [PGMC\\_CPC\\_CORE\\_PowerOffByCpuPowerMode](#) (PGMC\_CPC\_Type \*base, [pgmc\\_cpu\\_mode\\_t](#) mode)  
*Powers off the CPC core module by the target CPU power mode, such as Wait mode.*
- static void [PGMC\\_CPC\\_CORE\\_PowerOffBySoftwareMode](#) (PGMC\_CPC\_Type \*base, bool powerOff)  
*Powers off/on the CPC core module by the software.*
- static void [PGMC\\_CPC\\_CORE\\_DisableLowPower](#) (PGMC\_CPC\_Type \*base)  
*Disables low power mode control, the CPU core will not be affected by any low power modes.*
- void [PGMC\\_CPC\\_CACHE\\_ControlByCpuPowerMode](#) (PGMC\_CPC\_Type \*base, [pgmc\\_cpu\\_mode\\_t](#) mode, [pgmc\\_memory\\_low\\_power\\_level\\_t](#) memoryLowPowerLevel)  
*Makes the CPC CACHE module controlled by the target CPU power mode, such as Wait mode.*
- void [PGMC\\_CPC\\_CACHE\\_ControlBySetPointMode](#) (PGMC\_CPC\_Type \*base, uint32\_t set-PointMap, [pgmc\\_memory\\_low\\_power\\_level\\_t](#) memoryLowPowerLevel)  
*Makes the CPC CACHE module controlled by the target set points.*
- static void [PGMC\\_CPC\\_CACHE\\_DisableLowPower](#) (PGMC\_CPC\_Type \*base)  
*Disables low power mode control, so the cache will not be affected by any low power modes.*
- void [PGMC\\_CPC\\_CACHE\\_TriggerMLPLSoftwareChange](#) (PGMC\_CPC\_Type \*base)  
*Requests CPC cache module's memory low power level change by software mode.*

- void [PGMC\\_CPC\\_LMEM\\_ControlByCpuPowerMode](#) (PGMC\_CPC\_Type \*base, [pgmc\\_cpu\\_mode\\_t](#) mode, [pgmc\\_memory\\_low\\_power\\_level\\_t](#) memoryLowPowerLevel)  
*Makes the CPC LMEM module controlled by the target CPU power mode, such as Wait mode.*
- void [PGMC\\_CPC\\_LMEM\\_ControlBySetPointMode](#) (PGMC\_CPC\_Type \*base, uint32\_t setPointMap, [pgmc\\_memory\\_low\\_power\\_level\\_t](#) memoryLowPowerLevel)  
*Makes the CPC LMEM module controlled by the target set points.*
- static void [PGMC\\_CPC\\_LMEM\\_DisableLowPower](#) (PGMC\_CPC\_Type \*base)  
*Disables low power mode control, so that the CPC LMEM will not be affected by any low power modes.*
- void [PGMC\\_CPC\\_LMEM\\_TriggerMLPLSoftwareChange](#) (PGMC\_CPC\_Type \*base)  
*Requests CPC LMEM module's memory low power level change in software mode.*
- static void [PGMC\\_CPC-AllowUserModeAccess](#) (PGMC\_CPC\_Type \*base, bool enable)  
*Allows user mode access or not for the CPC module.*
- static void [PGMC\\_CPC-AllowNonSecureModeAccess](#) (PGMC\_CPC\_Type \*base, bool enable)  
*Allows non-secure mode access or not for the CPC module.*
- static void [PGMC\\_CPC\\_LockAccessSetting](#) (PGMC\_CPC\_Type \*base)  
*Locks access related settings, including secure access setting and user access setting, for the CPC module.*
- static void [PGMC\\_CPC\\_SetDomainIdWhiteList](#) (PGMC\_CPC\_Type \*base, uint8\_t domainId)  
*Sets the corresponding domain ID that can access CPU mode control registers for the CPC module.*
- static void [PGMC\\_CPC\\_LockDomainIDWhiteList](#) (PGMC\_CPC\_Type \*base)  
*Locks the value of Domain ID white list for CPC module.*
- static void [PGMC\\_CPC\\_LockLowPowerConfigurationFields](#) (PGMC\_CPC\_Type \*base)  
*Locks CPC realted low power configuration fields for CPC module.*

## MIF Module Related APIs

- void [PGMC\\_MIF\\_SetSignalBehaviour](#) (PGMC\_MIF\_Type \*base, [pgmc\\_memory\\_low\\_power\\_level\\_t](#) memoryLevel, uint32\_t mask)  
*Sets the behaviour of each signal in MIF, such as Sleep signal.*
- static void [PGMC\\_MIF\\_LockLowPowerConfigurationFields](#) (PGMC\_MIF\_Type \*base)  
*Locks MIF realted low power configuration fields for MIF module.*

## PMIC Power Related Interfaces

- static void [PGMC\\_PPC\\_TriggerPMICStandbySoftMode](#) (PGMC\_PPC\_Type \*base, bool enable)  
*Trigger PMIC standby ON/OFF.*
- void [PGMC\\_PPC\\_ControlByCpuPowerMode](#) (PGMC\_PPC\_Type \*base, [pgmc\\_cpu\\_mode\\_t](#) mode)  
*Makes the PMIC module controlled by the target CPU power mode, such as Wait mode.*
- void [PGMC\\_PPC\\_ControlBySetPointMode](#) (PGMC\_PPC\_Type \*base, uint32\_t setPointMap, bool enableStandby)  
*Makes the PMIC module controlled by the target set points.*
- static void [PGMC\\_PPC\\_DisableLowPower](#) (PGMC\_PPC\_Type \*base)  
*Disables low power mode control.*
- static void [PGMC\\_PPC-AllowUserModeAccess](#) (PGMC\_PPC\_Type \*base, bool enable)  
*Allows user mode access or not for PMIC module.*
- static void [PGMC\\_PPC-AllowNonSecureModeAccess](#) (PGMC\_PPC\_Type \*base, bool enable)  
*Allows non-secure mode access or not for the PMIC module.*
- static void [PGMC\\_PPC\\_LockAccessSetting](#) (PGMC\_PPC\_Type \*base)  
*Locks access related settings, including secure access setting and user access setting, for the PMIC module.*

- static void [PGMC\\_PPC\\_SetDomainIdWhiteList](#) (PGMC\_PPC\_Type \*base, uint8\_t domainId)  
*Sets the corresponding domain ID that can access CPU mode control registers for the PMIC module.*
- static void [PGMC\\_PPC\\_LockDomainIDWhiteList](#) (PGMC\_PPC\_Type \*base)  
*Locks the value of Domain ID white list for the PMIC module.*
- static void [PGMC\\_PPC\\_LockLowPowerConfigurationFields](#) (PGMC\_PPC\_Type \*base)  
*Locks low power configuration fields for the PMIC module.*

## 78.2 Data Structure Documentation

### 78.2.1 struct pgmc\_bpc\_cpu\_power\_mode\_option\_t

#### Data Fields

- [pgmc\\_bpc\\_assign\\_domain\\_t assignDomain](#)  
*Domain assignment of the BPC.*
- bool [stateSave](#)  
*Request save the state of power domain before entering target power mode.*
- bool [powerOff](#)  
*Request power off the power domain.*

#### Field Documentation

#### (1) pgmc\_bpc\_assign\_domain\_t pgmc\_bpc\_cpu\_power\_mode\_option\_t::assignDomain

The power mode of the selected core domain will control the selected power domain.

#### (2) bool pgmc\_bpc\_cpu\_power\_mode\_option\_t::stateSave

- **true** Save data when domain enter the selected mode.
- **false** Do not save data when domain enter the selected mode.

#### (3) bool pgmc\_bpc\_cpu\_power\_mode\_option\_t::powerOff

- **true** Power off the power domain when enter the selected mode.
- **false** Do not power off the power domain when enter the selected mode.

### 78.2.2 struct pgmc\_bpc\_setpoint\_mode\_option\_t

#### Data Fields

- bool [stateSave](#)  
*Request save the state of power domain before entering target setpoint.*
- bool [powerOff](#)  
*Request power off the power domain.*

#### Field Documentation

**(1) bool pgmc\_bpc\_setpoint\_mode\_option\_t::stateSave**

- **true** Save data when domain enter the selected setpoint.
- **false** Do not save data when domain enter the selected setpoint.

**(2) bool pgmc\_bpc\_setpoint\_mode\_option\_t::powerOff**

- **true** Power off the power domain when enter the selected setpoint.
- **false** Do not power off the power domain when enter the selected setpoint.

**78.3 Macro Definition Documentation****78.3.1 #define FSL\_PGMC\_RIVER\_VERSION (MAKE\_VERSION(2, 1, 1))****78.4 Enumeration Type Documentation****78.4.1 anonymous enum**

Enumerator

*kPGMC\_SetPoint0* The mask of set point0.  
*kPGMC\_SetPoint1* The mask of set point1.  
*kPGMC\_SetPoint2* The mask of set point2.  
*kPGMC\_SetPoint3* The mask of set point3.  
*kPGMC\_SetPoint4* The mask of set point4.  
*kPGMC\_SetPoint5* The mask of set point5.  
*kPGMC\_SetPoint6* The mask of set point6.  
*kPGMC\_SetPoint7* The mask of set point7.  
*kPGMC\_SetPoint8* The mask of set point8.  
*kPGMC\_SetPoint9* The mask of set point9.  
*kPGMC\_SetPoint10* The mask of set point10.  
*kPGMC\_SetPoint11* The mask of set point11.  
*kPGMC\_SetPoint12* The mask of set point12.  
*kPGMC\_SetPoint13* The mask of set point13.  
*kPGMC\_SetPoint14* The mask of set point14.  
*kPGMC\_SetPoint15* The mask of set point15.

**78.4.2 enum \_pgmc\_mif\_signal\_behaviour**

Enumerator

*kPGMC\_AssertSleepSignal* Assert Sleep signal.  
*kPGMC\_AssertInputGateSignal* Assert InputGate signal.  
*kPGMC\_AssertLowSpeedSignal* Assert LowSpeed signal.  
*kPGMC\_AssertHighSpeedSignal* Assert HighSpeed signal.

***kPGMC\_AssertStandbySignal*** Assert Standby signal.  
***kPGMC\_AssertArrayPowerDownSignal*** Assert ArrayPowerDown signal.  
***kPGMC\_AssertPeripheralPowerDownSignal*** Assert PeripheralPowerDown signal.  
***kPGMC\_AssertInitnSignal*** Assert Initn signal.  
***kPGMC\_AssertSwitch1OffSignal*** Assert Switch1Off signal.  
***kPGMC\_AssertSwitch2OffSignal*** Assert Switch2Off signal.  
***kPGMC\_AssertIsoSignal*** Assert Iso\_en signal.

### 78.4.3 enum pgmc\_bpc\_assign\_domain\_t

Enumerator

***kPGMC\_CM7Core*** CM7 Core domain.  
***kPGMC\_CM4Core*** CM4 Core domain.

### 78.4.4 enum pgmc\_cpu\_mode\_t

Enumerator

***kPGMC\_RunMode*** RUN mode.  
***kPGMC\_WaitMode*** WAIT mode.  
***kPGMC\_StopMode*** STOP mode.  
***kPGMC\_SuspendMode*** SUSPEND mode.

### 78.4.5 enum pgmc\_control\_mode\_t

### 78.4.6 enum pgmc\_memory\_low\_power\_level\_t

Enumerator

***kPGMC\_MLPLHighSpeed*** Memory low power level: High speed.  
***kPGMC\_MLPLNormal*** Memory low power level: Normal.  
***kPGMC\_MLPLLowSpeed*** Memory low power level: Low Speed.  
***kPGMC\_MLPLInputGating*** Memory low power level: Input Gating.  
***kPGMC\_MLPLStandby*** Memory low power level: Standby.  
***kPGMC\_MLPLSleep*** Memory low power level: Sleep.  
***kPGMC\_MLPLArrOnPerOff*** Memory low power level: Arr on per off.  
***kPGMC\_MLPLArrOffPerOn*** Memory low power level: Arr off per on.  
***kPGMC\_MLPLArrOffPerOff*** Memory low power level: Arr off per off.  
***kPGMC\_MLPLSw2*** Memory low power level: SW2.  
***kPGMC\_MLPLSw2PerOff*** Memory low power level: SW2 Per off.  
***kPGMC\_MLPLSw1PerOff*** Memory low power level: SW1 Per off.

## 78.4.7 enum pgmc\_mif\_signal\_t

Enumerator

*kPGMC\_SleepSignal* MIF Sleep signal.  
*kPGMC\_InputGateSignal* MIF InputGate signal.  
*kPGMC\_LowSpeedSignal* MIF LowSpeed signal.  
*kPGMC\_HighSpeedSignal* MIF HighSpeed signal.  
*kPGMC\_StandbySignal* MIF Standby signal.  
*kPGMC\_ArrayPowerDownSignal* MIF ArrayPowerDown signal.  
*kPGMC\_PeripheralPowerDownSignal* MIF PeripheralPowerDown signal.  
*kPGMC\_InitnSignal* MIF Initn signal.  
*kPGMC\_Switch1OffSignal* MIF Switch1Off signal.  
*kPGMC\_Switch2OffSignal* MIF Switch2Off signal.  
*kPGMC\_IsoSignal* MIF Iso\_en signal.

## 78.5 Function Documentation

**78.5.1 void PGMC\_BPC\_ControlPowerDomainByCpuPowerMode ( PGMC\_BPC\_Type \* *base*, pgmc\_cpu\_mode\_t *mode*, const pgmc\_bpc\_cpu\_power\_mode\_option\_t \* *option* )**

This function makes the module controlled by four typical CPU power modes, It also configs the resource domain and set memory low power level.

Parameters

|               |                                                                            |
|---------------|----------------------------------------------------------------------------|
| <i>base</i>   | PGMC basic power controller base address.                                  |
| <i>mode</i>   | Target CPU power mode.                                                     |
| <i>option</i> | The pointer of <a href="#">pgmc_bpc_cpu_power_mode_option_t</a> structure. |

**78.5.2 void PGMC\_BPC\_ControlPowerDomainBySetPointMode ( PGMC\_BPC\_Type \* *base*, uint32\_t *setPointMap*, const pgmc\_bpc\_setpoint\_mode\_option\_t \* *option* )**

This function makes the module controlled by specific set point, It also supports set memory low power level.

Note

When setting more than one set point, use "|" between the map values in [\\_pgmc\\_setpoint\\_map](#).

## Parameters

|                    |                                                                        |
|--------------------|------------------------------------------------------------------------|
| <i>base</i>        | PGMC basic power controller base address.                              |
| <i>setPointMap</i> | Should be the OR'ed value of <code>_pgmc_setpoint_map</code> .         |
| <i>option</i>      | The pointer of <code>pgmc_bpc_setpoint_mode_option_t</code> structure. |

### 78.5.3 void PGMC\_BPC\_ControlPowerDomainBySoftwareMode ( PGMC\_BPC\_Type \* *base*, bool *powerOff* )

## Note

The function is used to control power domain when the CPU is in RUN mode.

## Parameters

|                 |                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>     | PGMC basic power controller base address.                                                                                                                                                                                    |
| <i>powerOff</i> | Power On/Off power domain in software mode. <ul style="list-style-type: none"> <li>• <b>true</b> Power off the power domain in software mode.</li> <li>• <b>false</b> Power on the power domain in software mode.</li> </ul> |

### 78.5.4 static void PGMC\_BPC\_DisableLowPower ( PGMC\_BPC\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | PGMC basic power controller base address. |
|-------------|-------------------------------------------|

### 78.5.5 static void PGMC\_BPC\_RequestStateRestoreAtRunMode ( PGMC\_BPC\_Type \* *base* ) [inline], [static]

## Parameters



|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | PGMC basic power controller base address. |
|-------------|-------------------------------------------|

**78.5.6 static void PGMC\_BPC\_RequestStateRestoreAtSetPoint ( PGMC\_BPC\_Type \* *base*, uint32\_t *setPointMap* ) [inline], [static]**

Note

When setting more than one set point, use "|" between the map values in [\\_pgmc\\_setpoint\\_map](#).

Parameters

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <i>base</i>        | PGMC basic power controller base address.                         |
| <i>setPointMap</i> | Should be the OR'ed value of <a href="#">_pgmc_setpoint_map</a> . |

**78.5.7 static void PGMC\_BPC\_AllowUserModeAccess ( PGMC\_BPC\_Type \* *base*, bool *enable* ) [inline], [static]**

Note

If locked access related settings, the setting via this function is useless.

Parameters

|               |                                                                                                                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PGMC basic power controller base address.                                                                                                                                                                                                                                        |
| <i>enable</i> | Used to control whether allow user mode access. <ul style="list-style-type: none"> <li>• <b>true</b> Allow both privilege and user mode to access CPU mode control registers.</li> <li>• <b>false</b> Allow only privilege mode to access CPU mode control registers.</li> </ul> |

**78.5.8 static void PGMC\_BPC\_AllowNonSecureModeAccess ( PGMC\_BPC\_Type \* *base*, bool *enable* ) [inline], [static]**

Note

If locked access related settings, the setting via this function is useless.

## Parameters

|               |                                                                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PGMC basic power controller base address.                                                                                                                                                                                                                                                                            |
| <i>enable</i> | Used to control whether allow non-secure mode to access CPU mode control registers. <ul style="list-style-type: none"> <li>• <b>true</b> Allow both secure and non-secure mode to access CPU mode control registers.</li> <li>• <b>false</b> Allow only secure mode to access CPU mode control registers.</li> </ul> |

### 78.5.9 static void PGMC\_BPC\_LockAccessSetting ( PGMC\_BPC\_Type \* *base* ) [inline], [static]

## Note

This function used to lock access related settings. After locked the related bit field can not be written unless POR.

## Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | PGMC basic power controller base address. |
|-------------|-------------------------------------------|

### 78.5.10 static void PGMC\_BPC\_SetDomainIdWhiteList ( PGMC\_BPC\_Type \* *base*, uint8\_t *domainId* ) [inline], [static]

## Note

If locked the domain ID white list, the setting via this function is useless.

## Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>base</i>     | PGMC basic power controller base address.                               |
| <i>domainId</i> | Should be the OR'ed value of <a href="#">pgmc_bpc_assign_domain_t</a> . |

### 78.5.11 static void PGMC\_BPC\_LockDomainIDWhiteList ( PGMC\_BPC\_Type \* *base* ) [inline], [static]

## Note

After locked the domain ID white list can not be written again unless POR.

## Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | PGMC basic power controller base address. |
|-------------|-------------------------------------------|

**78.5.12 static void PGMC\_BPC\_LockLowPowerConfigurationFields ( PGMC\_BPC\_Type \* *base* ) [inline], [static]**

## Note

After locked the low power configurations fields can not be updated unless POR.

## Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | PGMC basic power controller base address. |
|-------------|-------------------------------------------|

**78.5.13 void PGMC\_CPC\_CORE\_PowerOffByCpuPowerMode ( PGMC\_CPC\_Type \* *base*, pgmc\_cpu\_mode\_t *mode* )**

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC CORE module base address. |
| <i>mode</i> | Target CPU power mode.        |

**78.5.14 static void PGMC\_CPC\_CORE\_PowerOffBySoftwareMode ( PGMC\_CPC\_Type \* *base*, bool *powerOff* ) [inline], [static]**

## Parameters

|                 |                                                                                                                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>     | CPC CORE module base address.                                                                                                                                                                  |
| <i>powerOff</i> | Used to power off/on the CPC core module. <ul style="list-style-type: none"> <li>• <b>true</b> Power off the CPC core module.</li> <li>• <b>false</b> Power on the CPC core module.</li> </ul> |

**78.5.15 static void PGMC\_CPC\_CORE\_DisableLowPower ( PGMC\_CPC\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC CORE module base address. |
|-------------|-------------------------------|

**78.5.16 void PGMC\_CPC\_CACHE\_ControlByCpuPowerMode ( PGMC\_CPC\_Type \* *base*, pgmc\_cpu\_mode\_t *mode*, pgmc\_memory\_low\_power\_level\_t *memoryLowPowerLevel* )**

This function makes the module controlled by four typical CPU power modes, it also can set memory low power level.

## Parameters

|                             |                                |
|-----------------------------|--------------------------------|
| <i>base</i>                 | CPC CACHE module base address. |
| <i>mode</i>                 | Target CPU power mode.         |
| <i>memoryLow-PowerLevel</i> | Memory low power level.        |

**78.5.17 void PGMC\_CPC\_CACHE\_ControlBySetPointMode ( PGMC\_CPC\_Type \* *base*, uint32\_t *setPointMap*, pgmc\_memory\_low\_power\_level\_t *memoryLowPowerLevel* )**

This function makes the module controlled by specific set point, It also supports set memory low power level.

## Note

When setting more than one set point, use "|" between the map values in [\\_pgmc\\_setpoint\\_map](#).

## Parameters

|                             |                                                                   |
|-----------------------------|-------------------------------------------------------------------|
| <i>base</i>                 | CPC CACHE module base address.                                    |
| <i>setPointMap</i>          | Should be the OR'ed value of <a href="#">_pgmc_setpoint_map</a> . |
| <i>memoryLow-PowerLevel</i> | Memory low power level.                                           |

**78.5.18 static void PGMC\_CPC\_CACHE\_DisableLowPower ( PGMC\_CPC\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | CPC CACHE module base address. |
|-------------|--------------------------------|

### 78.5.19 void PGMC\_CPC\_CACHE\_TriggerMLPLSoftwareChange ( PGMC\_CPC\_Type \* *base* )

## Note

If request memory low power level change, must wait the MLPL transition complete.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC LMEM module base address. |
|-------------|-------------------------------|

### 78.5.20 void PGMC\_CPC\_LMEM\_ControlByCpuPowerMode ( PGMC\_CPC\_Type \* *base*, pgmc\_cpu\_mode\_t *mode*, pgmc\_memory\_low\_power\_level\_t *memoryLowPowerLevel* )

This function makes the module controlled by four typical CPU power modes, it also can set memory low power level.

## Parameters

|                             |                               |
|-----------------------------|-------------------------------|
| <i>base</i>                 | CPC LMEM module base address. |
| <i>mode</i>                 | Target CPU power mode.        |
| <i>memoryLow-PowerLevel</i> | Memory low power level.       |

### 78.5.21 void PGMC\_CPC\_LMEM\_ControlBySetPointMode ( PGMC\_CPC\_Type \* *base*, uint32\_t *setPointMap*, pgmc\_memory\_low\_power\_level\_t *memoryLowPowerLevel* )

This function makes the module controlled by specific set point, It also supports set memory low power level.

## Note

When setting more than one set point, use "|" between the map values in [\\_pgmc\\_setpoint\\_map](#).

## Parameters

|                             |                                                                   |
|-----------------------------|-------------------------------------------------------------------|
| <i>base</i>                 | CPC LMEM module base address.                                     |
| <i>setPointMap</i>          | Should be the OR'ed value of <a href="#">_pgmc_setpoint_map</a> . |
| <i>memoryLow-PowerLevel</i> | Memory low power level.                                           |

**78.5.22 static void PGMC\_CPC\_LMEM\_DisableLowPower ( PGMC\_CPC\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC LMEM module base address. |
|-------------|-------------------------------|

**78.5.23 void PGMC\_CPC\_LMEM\_TriggerMLPLSoftwareChange ( PGMC\_CPC\_Type \* *base* )**

## Note

If request memory low power level change, must wait the MLPL transition complete.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC LMEM module base address. |
|-------------|-------------------------------|

**78.5.24 static void PGMC\_CPC\_AllowUserModeAccess ( PGMC\_CPC\_Type \* *base*, bool *enable* ) [inline], [static]**

## Note

If locked access related settings, the setting via this function is useless.

## Parameters

---

|               |                                                                                                                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | CPC LMEM module base address.                                                                                                                                                                                                                                                    |
| <i>enable</i> | Used to control whether allow user mode access. <ul style="list-style-type: none"> <li>• <b>true</b> Allow both privilege and user mode to access CPU mode control registers.</li> <li>• <b>false</b> Allow only privilege mode to access CPU mode control registers.</li> </ul> |

**78.5.25 static void PGMC\_CPC\_AllowNonSecureModeAccess ( PGMC\_CPC\_Type \* *base*, bool *enable* ) [inline], [static]**

Note

If locked access related settings, the setting via this function is useless.

Parameters

|               |                                                                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | CPC LMEM module base address.                                                                                                                                                                                                                                                                                        |
| <i>enable</i> | Used to control whether allow non-secure mode to access CPU mode control registers. <ul style="list-style-type: none"> <li>• <b>true</b> Allow both secure and non-secure mode to access CPU mode control registers.</li> <li>• <b>false</b> Allow only secure mode to access CPU mode control registers.</li> </ul> |

**78.5.26 static void PGMC\_CPC\_LockAccessSetting ( PGMC\_CPC\_Type \* *base* ) [inline], [static]**

Note

This function used to lock access related settings. After locked the related bit field can not be written unless POR.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC LMEM module base address. |
|-------------|-------------------------------|

**78.5.27 static void PGMC\_CPC\_SetDomainIdWhiteList ( PGMC\_CPC\_Type \* *base*, uint8\_t *domainId* ) [inline], [static]**

## Note

If the domain ID whitelist is locked, the setting via this function is useless.

## Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>base</i>     | CPC LMEM module base address.                                           |
| <i>domainId</i> | Should be the OR'ed value of <a href="#">pgmc_bpc_assign_domain_t</a> . |

### 78.5.28 static void PGMC\_CPC\_LockDomainIDWhiteList ( PGMC\_CPC\_Type \* *base* ) [inline], [static]

## Note

After locked the domain ID white list can not be written again unless POR.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC LMEM module base address. |
|-------------|-------------------------------|

### 78.5.29 static void PGMC\_CPC\_LockLowPowerConfigurationFields ( PGMC\_CPC\_Type \* *base* ) [inline], [static]

## Note

After locked the low power configurations fields can not be updated unless POR.

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | CPC LMEM module base address. |
|-------------|-------------------------------|

### 78.5.30 void PGMC\_MIF\_SetSignalBehaviour ( PGMC\_MIF\_Type \* *base*, pgmc\_memory\_low\_power\_level\_t *memoryLevel*, uint32\_t *mask* )

## Note

To control the memory low power operation, this function must be invoked after selecting the memory low power level. Use case:

```
*
* PGMC_BPC_ControlPowerDomainByCpuPowerMode (
PGMC_BPC0_BASE, kPGMC_WaitMode, kPGMC_CM7Core,
* kPGMC_MLPLSleep, false);
* PGMC_MIF_SetSignalBehaviour (PGMC_BPC0_MIF_BASE,
kPGMC_MLPLSleep, kPGMC_AssertSleepSignal);
*
```



## Parameters

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | PGMC MIF peripheral base address.                                                                                |
| <i>memoryLevel</i> | The selected memory low power level. For details please refer to <a href="#">pgmc_memory_low_power_level_t</a> . |
| <i>mask</i>        | The mask of MIF signal behaviour. Should be the OR'ed value of <a href="#">_pgmc_mif_signal_behaviour</a>        |

### 78.5.31 static void PGMC\_MIF\_LockLowPowerConfigurationFields ( PGMC\_MIF\_Type \* *base* ) [inline], [static]

## Note

After locked the low power configurations fields can not be updated unless POR.

## Parameters

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

### 78.5.32 static void PGMC\_PPC\_TriggerPMICStandbySoftMode ( PGMC\_PPC\_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"> <li>• <b>true</b> Trigger PMIC standby ON.</li> <li>• <b>false</b> Trigger PMIC standby OFF.</li> </ul> |

### 78.5.33 void PGMC\_PPC\_ControlByCpuPowerMode ( PGMC\_PPC\_Type \* *base*, pgmc\_cpu\_mode\_t *mode* )

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | PMIC module base address. |
| <i>mode</i> | Target CPU power mode.    |

### 78.5.34 void PGMC\_PPC\_ControlBySetPointMode ( PGMC\_PPC\_Type \* *base*, uint32\_t *setPointMap*, bool *enableStandby* )

This function makes the module controlled by specific set point, It also supports PMIC standby on.

## Note

When setting more than one set point, use "|" between the map values in [\\_pgmc\\_setpoint\\_map](#).

## Parameters

|                      |                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | PMIC module base address.                                                                                                                           |
| <i>setPointMap</i>   | Should be the OR'ed value of <a href="#">_pgmc_setpoint_map</a> .                                                                                   |
| <i>enableStandby</i> | true: PMIC standby on when system enters set point number and system is in standby mode. false: PMIC standby on when system enters set point number |

### 78.5.35 static void PGMC\_PPC\_DisableLowPower ( PGMC\_PPC\_Type \* *base* ) [inline], [static]

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | PMIC module bsase address. |
|-------------|----------------------------|

### 78.5.36 static void PGMC\_PPC\_AllowUserModeAccess ( PGMC\_PPC\_Type \* *base*, bool *enable* ) [inline], [static]

## Note

If locked access related settings, the setting via this function is useless.

## Parameters

|               |                                                                                                                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PMIC module base address.                                                                                                                                                                                                                                                        |
| <i>enable</i> | Used to control whether allow user mode access. <ul style="list-style-type: none"> <li>• <b>true</b> Allow both privilege and user mode to access CPU mode control registers.</li> <li>• <b>false</b> Allow only privilege mode to access CPU mode control registers.</li> </ul> |

**78.5.37 static void PGMC\_PPC\_AllowNonSecureModeAccess ( PGMC\_PPC\_Type \* *base*, bool *enable* ) [inline], [static]**

## Note

If locked access related settings, the setting via this function is useless.

## Parameters

|               |                                                                                                                                                                                                                                                                                                                      |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | PMIC module base address.                                                                                                                                                                                                                                                                                            |
| <i>enable</i> | Used to control whether allow non-secure mode to access CPU mode control registers. <ul style="list-style-type: none"> <li>• <b>true</b> Allow both secure and non-secure mode to access CPU mode control registers.</li> <li>• <b>false</b> Allow only secure mode to access CPU mode control registers.</li> </ul> |

**78.5.38 static void PGMC\_PPC\_LockAccessSetting ( PGMC\_PPC\_Type \* *base* ) [inline], [static]**

## Note

This function used to lock access related settings. After locked the related bit field can not be written unless POR.

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | PMIC module base address. |
|-------------|---------------------------|

**78.5.39 static void PGMC\_PPC\_SetDomainIdWhiteList ( PGMC\_PPC\_Type \* *base*, uint8\_t *domainId* ) [inline], [static]**

## Note

If the domain ID whitelist is locked, the setting via this function is useless.

## Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>base</i>     | PMIC module base address.                                               |
| <i>domainId</i> | Should be the OR'ed value of <a href="#">pgmc_bpc_assign_domain_t</a> . |

**78.5.40 static void PGMC\_PPC\_LockDomainIDWhiteList ( PGMC\_PPC\_Type \* *base* ) [inline], [static]**

## Note

After locked the domain ID white list can not be written again unless POR.

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | PMIC module base address. |
|-------------|---------------------------|

**78.5.41 static void PGMC\_PPC\_LockLowPowerConfigurationFields ( PGMC\_PPC\_Type \* *base* ) [inline], [static]**

## Note

After locked the low power configurations fields can not be updated unless POR.

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | PMIC module base address. |
|-------------|---------------------------|

# Chapter 79

## Soc\_mipi\_csi2rx

### 79.1 Overview

#### Files

- file [fsl\\_soc\\_mipi\\_csi2rx.h](#)

#### Macros

- #define [FSL\\_SOC\\_MIPI\\_CSI2RX\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 2))  
*Driver version.*

#### Functions

- void [MIPI\\_CSI2RX\\_SoftwareReset](#) (MIPI\_CSI2RX\_Type \*base, bool reset)  
*Assert or deassert CSI2RX reset in system level.*
- void [MIPI\\_CSI2RX\\_InitInterface](#) (MIPI\_CSI2RX\_Type \*base, uint8\_t tHsSettle\_EscClk)  
*Initialize the CSI2RX interface.*
- void [MIPI\\_CSI2RX\\_DeinitInterface](#) (MIPI\_CSI2RX\_Type \*base)  
*Deinitialize the CSI2RX interface.*

### 79.2 Macro Definition Documentation

**79.2.1 #define FSL\_SOC\_MIPI\_CSI2RX\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))**

### 79.3 Function Documentation

**79.3.1 void MIPI\_CSI2RX\_SoftwareReset ( MIPI\_CSI2RX\_Type \* *base*, bool *reset* )**

#### Parameters

|              |                                                             |
|--------------|-------------------------------------------------------------|
| <i>base</i>  | The CSI2RX peripheral base address.                         |
| <i>reset</i> | Pass in true to set to reset state, false to release reset. |

#### Note

Don't call this function directly.

**79.3.2 void MIPI\_CSI2RX\_InitInterface ( MIPI\_CSI2RX\_Type \* *base*, uint8\_t *tHsSettle\_EscClk* )**

## Parameters

|                          |                                     |
|--------------------------|-------------------------------------|
| <i>base</i>              | The CSI2RX peripheral base address. |
| <i>tHsSettle_Esc-Clk</i> | t-HS_SETTLE in esc clock period.    |

## Note

Don't call this function directly.

### 79.3.3 void MIPI\_CSI2RX\_DeinitInterface ( MIPI\_CSI2RX\_Type \* *base* )

## Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | The CSI2RX peripheral base address. |
|-------------|-------------------------------------|

## Note

Don't call this function directly.

## Chapter 80

# Asrc\_edma\_driver

### 80.1 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, 1, 0))  
*Version 2.1.0.*

#### 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\\_pair\\_config\\_t](#) \*asrcConfig, uint32\_t inSampleRate, uint32\_t outSampleRate)

- *Configures the ASRC P2P channel pair.*  
uint32\_t [ASRC\\_GetOutSamplesSizeEDMA](#) (ASRC\_Type \*base, asrc\_edma\_handle\_t \*handle, uint32\_t inSampleRate, uint32\_t outSampleRate, uint32\_t inSamplesize)
- *Get output sample buffer size can be transferred by edma.*  
status\_t [ASRC\\_TransferEDMA](#) (ASRC\_Type \*base, asrc\_edma\_handle\_t \*handle, [asrc\\_transfer\\_t](#) \*xfer)
- *Performs a non-blocking ASRC m2m convert using 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.*

## 80.2 Data Structure Documentation

### 80.2.1 struct asrc\_p2p\_edma\_config\_t

#### Data Fields

- uint8\_t [watermark](#)  
*peripheral watermark*
- uint8\_t [channel](#)  
*peripheral channel number*
- [asrc\\_start\\_peripheral\\_t](#) [startPeripheral](#)  
*trigger peripheral start*

### 80.2.2 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`

### 80.2.3 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`

## 80.2.4 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.*

## 80.3 Macro Definition Documentation

### 80.3.1 #define ASRC\_XFER\_IN\_QUEUE\_SIZE 4U

ASRC IN edma QUEUE size

## 80.4 Function Documentation

**80.4.1 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. |

**80.4.2 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. |

**80.4.3 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. |
|----------------------|--------------------------|

**80.4.4** `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. |
|---------------|----------------------|

**80.4.5** `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.      |

#### 80.4.6 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. |

#### 80.4.7 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. |

#### 80.4.8 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. |

#### 80.4.9 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 81

## Enet\_qos\_qos

### 81.1 Overview

#### Data Structures

- struct [enet\\_qos\\_rx\\_bd\\_struct\\_t](#)  
*Defines the receive descriptor structure has the read-format and write-back format structure. [More...](#)*
- struct [enet\\_qos\\_tx\\_bd\\_struct\\_t](#)  
*Defines the transmit descriptor structure has the read-format and write-back format structure. [More...](#)*
- struct [enet\\_qos\\_ptp\\_time\\_t](#)  
*Defines the ENET PTP time stamp structure. [More...](#)*
- struct [enet\\_qos\\_frame\\_info\\_t](#)  
*Defines the frame info structure. [More...](#)*
- struct [enet\\_qos\\_tx\\_dirty\\_ring\\_t](#)  
*Defines the ENET transmit dirty addresses ring/queue structure. [More...](#)*
- struct [enet\\_qos\\_ptp\\_config\\_t](#)  
*Defines the ENET PTP configuration structure. [More...](#)*
- struct [enet\\_qos\\_est\\_gate\\_op\\_t](#)  
*Defines the EST gate operation structure. [More...](#)*
- struct [enet\\_qos\\_est\\_gcl\\_t](#)  
*Defines the EST gate control list structure. [More...](#)*
- struct [enet\\_qos\\_rxp\\_config\\_t](#)  
*Defines the ENET\_QOS Rx parser configuration structure. [More...](#)*
- struct [enet\\_qos\\_buffer\\_config\\_t](#)  
*Defines the buffer descriptor configure structure. [More...](#)*
- struct [enet\\_qos\\_cbs\\_config\\_t](#)  
*Defines the CBS configuration for queue. [More...](#)*
- struct [enet\\_qos\\_queue\\_tx\\_config\\_t](#)  
*Defines the queue configuration structure. [More...](#)*
- struct [enet\\_qos\\_queue\\_rx\\_config\\_t](#)  
*Defines the queue configuration structure. [More...](#)*
- struct [enet\\_qos\\_multiqueue\\_config\\_t](#)  
*Defines the configuration when multi-queue is used. [More...](#)*
- struct [enet\\_qos\\_config\\_t](#)  
*Defines the basic configuration structure for the ENET device. [More...](#)*
- struct [enet\\_qos\\_tx\\_bd\\_ring\\_t](#)  
*Defines the ENET transmit buffer descriptor ring/queue structure. [More...](#)*
- struct [enet\\_qos\\_rx\\_bd\\_ring\\_t](#)  
*Defines the ENET receive buffer descriptor ring/queue structure. [More...](#)*
- struct [enet\\_qos\\_handle\\_t](#)  
*Defines the ENET handler structure. [More...](#)*
- struct [enet\\_qos\\_buffer\\_struct\\_t](#)  
*Defines the frame buffer structure. [More...](#)*
- struct [enet\\_qos\\_rx\\_frame\\_error\\_t](#)  
*Defines the Rx frame error structure. [More...](#)*
- struct [enet\\_qos\\_rx\\_frame\\_struct\\_t](#)

- Defines the Rx frame data structure. [More...](#)
- struct [enet\\_qos\\_transfer\\_stats\\_t](#)  
Defines the ENET QOS transfer statistics structure. [More...](#)

## Typedefs

- typedef void (\* [enet\\_qos\\_rx\\_alloc\\_callback\\_t](#))(ENET\_QOS\_Type \*base, void \*userData, uint8\_t channel)  
Defines the Rx memory buffer alloc function pointer.
- typedef void (\* [enet\\_qos\\_rx\\_free\\_callback\\_t](#))(ENET\_QOS\_Type \*base, void \*buffer, void \*userData, uint8\_t channel)  
Defines the Rx memory buffer free function pointer.
- typedef void (\* [enet\\_qos\\_callback\\_t](#))(ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, [enet\\_qos\\_event\\_t](#) event, uint8\_t channel, void \*userData)  
ENET callback function.

## Enumerations

- enum {  
[kStatus\\_ENET\\_QOS\\_InitMemoryFail](#),  
[kStatus\\_ENET\\_QOS\\_RxFrameError](#),  
[kStatus\\_ENET\\_QOS\\_RxFrameFail](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 2U),  
[kStatus\\_ENET\\_QOS\\_RxFrameEmpty](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 3U),  
[kStatus\\_ENET\\_QOS\\_RxFrameDrop](#),  
[kStatus\\_ENET\\_QOS\\_TxFrameBusy](#),  
[kStatus\\_ENET\\_QOS\\_TxFrameFail](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 6U),  
[kStatus\\_ENET\\_QOS\\_TxFrameOverLen](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 7U),  
[kStatus\\_ENET\\_QOS\\_Est\\_SwListBusy](#),  
[kStatus\\_ENET\\_QOS\\_Est\\_SwListWriteAbort](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 9-U),  
[kStatus\\_ENET\\_QOS\\_Est\\_InvalidParameter](#),  
[kStatus\\_ENET\\_QOS\\_Est\\_BtrError](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 11U),  
[kStatus\\_ENET\\_QOS\\_TrgtBusy](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 12U),  
[kStatus\\_ENET\\_QOS\\_Timeout](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 13U),  
[kStatus\\_ENET\\_QOS\\_PpsBusy](#) = MAKE\_STATUS(kStatusGroup\_ENET\_QOS, 14U) }  
Defines the status return codes for transaction.
- enum [enet\\_qos\\_mii\\_mode\\_t](#) {  
[kENET\\_QOS\\_MiiMode](#) = 0U,  
[kENET\\_QOS\\_RgmiiMode](#) = 1U,  
[kENET\\_QOS\\_RmiiMode](#) = 4U }  
Defines the MII/RGMII mode for data interface between the MAC and the PHY.
- enum [enet\\_qos\\_mii\\_speed\\_t](#) {  
[kENET\\_QOS\\_MiiSpeed10M](#),  
[kENET\\_QOS\\_MiiSpeed100M](#),  
[kENET\\_QOS\\_MiiSpeed1000M](#),  
[kENET\\_QOS\\_MiiSpeed2500M](#) }  
Defines the 10/100/1000 Mbps speed for the MII data interface.



- enum `enet_qos_mii_duplex_t` {  
`kENET_QOS_MiiHalfDuplex` = 0U,  
`kENET_QOS_MiiFullDuplex` }  
*Defines the half or full duplex for the MII data interface.*
- enum `enet_qos_mii_normal_opcode` {  
`kENET_QOS_MiiWriteFrame`,  
`kENET_QOS_MiiReadFrame` }  
*Define the MII opcode for normal MDIO\_CLAUSES\_22 Frame.*
- enum `enet_qos_dma_burstlen` {  
`kENET_QOS_BurstLen1` = 0x00001U,  
`kENET_QOS_BurstLen2` = 0x00002U,  
`kENET_QOS_BurstLen4` = 0x00004U,  
`kENET_QOS_BurstLen8` = 0x00008U,  
`kENET_QOS_BurstLen16` = 0x00010U,  
`kENET_QOS_BurstLen32` = 0x00020U,  
`kENET_QOS_BurstLen64` = 0x00040U,  
`kENET_QOS_BurstLen128` = 0x00080U,  
`kENET_QOS_BurstLen256` = 0x00100U }  
*Define the DMA maximum transmit burst length.*
- enum `enet_qos_desc_flag` {  
`kENET_QOS_MiddleFlag` = 0,  
`kENET_QOS_LastFlagOnly`,  
`kENET_QOS_FirstFlagOnly`,  
`kENET_QOS_FirstLastFlag` }  
*Define the flag for the descriptor.*
- enum `enet_qos_systime_op` {  
`kENET_QOS_SystimeAdd` = 0U,  
`kENET_QOS_SystimeSubtract` = 1U }  
*Define the system time adjust operation control.*
- enum `enet_qos_ts_rollover_type` {  
`kENET_QOS_BinaryRollover` = 0,  
`kENET_QOS_DigitalRollover` = 1 }  
*Define the system time rollover control.*
- enum `enet_qos_special_config_t` {  
`kENET_QOS_DescDoubleBuffer` = 0x0001U,  
`kENET_QOS_StoreAndForward` = 0x0002U,  
`kENET_QOS_PromiscuousEnable` = 0x0004U,  
`kENET_QOS_FlowControlEnable` = 0x0008U,  
`kENET_QOS_BroadCastRxDisable` = 0x0010U,  
`kENET_QOS_MulticastAllEnable` = 0x0020U,  
`kENET_QOS_8023AS2KPacket` = 0x0040U,  
`kENET_QOS_HashMulticastEnable` = 0x0080U }  
*Defines some special configuration for ENET.*
- enum `enet_qos_dma_interrupt_enable_t` {

```

kENET_QOS_DmaTx = ENET_QOS_DMA_CHX_INT_EN_TIE_MASK,
kENET_QOS_DmaTxStop = ENET_QOS_DMA_CHX_INT_EN_TXSE_MASK,
kENET_QOS_DmaTxBuffUnavail = ENET_QOS_DMA_CHX_INT_EN_TBUE_MASK,
kENET_QOS_DmaRx = ENET_QOS_DMA_CHX_INT_EN_RIE_MASK,
kENET_QOS_DmaRxBuffUnavail = ENET_QOS_DMA_CHX_INT_EN_RBUE_MASK,
kENET_QOS_DmaRxStop = ENET_QOS_DMA_CHX_INT_EN_RSE_MASK,
kENET_QOS_DmaRxWatchdogTimeout = ENET_QOS_DMA_CHX_INT_EN_RWTE_MASK,
kENET_QOS_DmaEarlyTx = ENET_QOS_DMA_CHX_INT_EN_ETIE_MASK,
kENET_QOS_DmaEarlyRx = ENET_QOS_DMA_CHX_INT_EN_ERIE_MASK,
kENET_QOS_DmaBusErr = ENET_QOS_DMA_CHX_INT_EN_FBEE_MASK }

```

*List of DMA interrupts supported by the ENET interrupt.*

- enum `enet_qos_mac_interrupt_enable_t`

*List of mac interrupts supported by the ENET interrupt.*

- enum `enet_qos_event_t` {  
`kENET_QOS_RxIntEvent`,  
`kENET_QOS_TxIntEvent`,  
`kENET_QOS_WakeUpIntEvent`,  
`kENET_QOS_TimeStampIntEvent` }

*Defines the common interrupt event for callback use.*

- enum `enet_qos_queue_mode_t` {  
`kENET_QOS_AVB_Mode` = 1U,  
`kENET_QOS_DCB_Mode` = 2U }

*Define the MTL mode for multiple queues/rings.*

- enum `enet_qos_mtl_multiqueue_txsche` {  
`kENET_QOS_txWeightRR` = 0U,  
`kENET_QOS_txWeightFQ` = 1U,  
`kENET_QOS_txDefictWeightRR` = 2U,  
`kENET_QOS_txStrPrio` = 3U }

*Define the MTL tx scheduling algorithm for multiple queues/rings.*

- enum `enet_qos_mtl_multiqueue_rxsche` {  
`kENET_QOS_rxStrPrio` = 0U,  
`kENET_QOS_rxWeightStrPrio` }

*Define the MTL rx scheduling algorithm for multiple queues/rings.*

- enum `enet_qos_mtl_rxqueuemap_t` {  
`kENET_QOS_StaticDirctMap` = 0x100U,  
`kENET_QOS_DynamicMap` }

*Define the MTL rx queue and DMA channel mapping.*

- enum `enet_qos_rx_queue_route_t`

*Defines the package type for receive queue routing.*

- enum `enet_qos_ptp_event_type_t` {  
`kENET_QOS_PtpEventMsgType` = 3U,  
`kENET_QOS_PtpSrcPortIdLen` = 10U,  
`kENET_QOS_PtpEventPort` = 319U,  
`kENET_QOS_PtpGnrlPort` = 320U }

*Defines the ENET PTP message related constant.*

- enum `enet_qos_ptp_pps_instance_t` {

```
kENET_QOS_PtpPpsInstance0 = 0U,
kENET_QOS_PtpPpsInstance1,
kENET_QOS_PtpPpsInstance2,
kENET_QOS_PtpPpsInstance3 }
```

*Defines the PPS instance numbers.*

- enum `enet_qos_ptp_pps_trgt_mode_t` {  
`kENET_QOS_PtpPpsTrgtModeOnlyInt` = 0U,  
`kENET_QOS_PtpPpsTrgtModeIntSt` = 2,  
`kENET_QOS_PtpPpsTrgtModeOnlySt` = 3 }

*Defines the Target Time register mode.*

- enum `enet_qos_ptp_pps_cmd_t` {  
`kENET_QOS_PtpPpsCmdNC` = 0U,  
`kENET_QOS_PtpPpsCmdSSP` = 1U,  
`kENET_QOS_PtpPpsCmdSPT` = 2U,  
`kENET_QOS_PtpPpsCmdCS` = 3U,  
`kENET_QOS_PtpPpsCmdSPTAT` = 4U,  
`kENET_QOS_PtpPpsCmdSPTI` = 5U,  
`kENET_QOS_PtpPpsCmdCSPT` = 6U }

*Defines commands for ppscmd register.*

- enum `enet_qos_ets_list_length_t` {  
`kENET_QOS_Ets_List_64` = 7U,  
`kENET_QOS_Ets_List_128` = 8U,  
`kENET_QOS_Ets_List_256` = 9U,  
`kENET_QOS_Ets_List_512` = 10U,  
`kENET_QOS_Ets_List_1024` = 11U }

*Defines the enumeration of ETS list length.*

- enum `enet_qos_ets_gccr_addr_t` {  
`kENET_QOS_Ets_btr_low` = 0U,  
`kENET_QOS_Ets_btr_high` = 1U,  
`kENET_QOS_Ets_ctr_low` = 2U,  
`kENET_QOS_Ets_ctr_high` = 3U,  
`kENET_QOS_Ets_ter` = 4U,  
`kENET_QOS_Ets_llr` = 5U }

*Defines the enumeration of ETS gate control address.*

- enum `enet_qos_rxp_dma_chn_t` {  
`kENET_QOS_Rxp_DMACHn0` = 1U,  
`kENET_QOS_Rxp_DMACHn1` = 2U,  
`kENET_QOS_Rxp_DMACHn2` = 4U,  
`kENET_QOS_Rxp_DMACHn3` = 8U,  
`kENET_QOS_Rxp_DMACHn4` = 16U }

*Defines the enumeration of DMA channel used for rx parser entry.*

## Functions

- void `ENET_QOS_SetSYSControl` (`enet_qos_mii_mode_t` miiMode)  
*Set ENET system configuration.*

## Variables

- const clock\_ip\_name\_t s\_enetqosClock []  
*Pointers to enet clocks for each instance.*

## Driver version

- #define FSL\_ENET\_QOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))  
*Defines the driver version.*

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

- #define ENET\_QOS\_RXDESCRIP\_RD\_BUFF1VALID\_MASK (1UL << 24U)  
*Defines for read format.*
- #define ENET\_QOS\_RXDESCRIP\_RD\_BUFF2VALID\_MASK (1UL << 25U)  
*Buffer2 address valid.*
- #define ENET\_QOS\_RXDESCRIP\_RD\_IOC\_MASK (1UL << 30U)  
*Interrupt enable on complete.*
- #define ENET\_QOS\_RXDESCRIP\_RD\_OWN\_MASK (1UL << 31U)  
*Own bit.*
- #define ENET\_QOS\_RXDESCRIP\_WR\_ERR\_MASK ((1UL << 3U) | (1UL << 7U))  
*Defines for write back format.*
- #define ENET\_QOS\_RXDESCRIP\_WR\_PYLOAD\_MASK (0x7UL)
- #define ENET\_QOS\_RXDESCRIP\_WR\_PTPMSGTYPE\_MASK (0xF00UL)
- #define ENET\_QOS\_RXDESCRIP\_WR\_PTPTYPE\_MASK (1UL << 12U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_PTPVERSION\_MASK (1UL << 13U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_PTPTSA\_MASK (1UL << 14U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_PACKETLEN\_MASK (0x7FFFUL)
- #define ENET\_QOS\_RXDESCRIP\_WR\_ERRSUM\_MASK (1UL << 15U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_TYPE\_MASK (0x30000UL)
- #define ENET\_QOS\_RXDESCRIP\_WR\_DE\_MASK (1UL << 19U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_RE\_MASK (1UL << 20U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_OE\_MASK (1UL << 21U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_RWT\_MASK (1UL << 22U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_GP\_MASK (1UL << 22U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_CRC\_MASK (1UL << 23U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_RS0V\_MASK (1UL << 25U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_RS1V\_MASK (1UL << 26U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_RS2V\_MASK (1UL << 27U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_LD\_MASK (1UL << 28U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_FD\_MASK (1UL << 29U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_CTXT\_MASK (1UL << 30U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_OWN\_MASK (1UL << 31U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_SA\_FAILURE\_MASK (1UL << 16U)
- #define ENET\_QOS\_RXDESCRIP\_WR\_DA\_FAILURE\_MASK (1UL << 17U)

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

- #define ENET\_QOS\_TXDESCRIP\_RD\_BL1\_MASK (0x3fffUL)  
*Defines for read format.*
- #define ENET\_QOS\_TXDESCRIP\_RD\_BL2\_MASK (ENET\_QOS\_TXDESCRIP\_RD\_BL1\_MASK << 16U)

- #define **ENET\_QOS\_TXDESCRIP\_RD\_BL1**(n) (((uint32\_t)(n)&**ENET\_QOS\_TXDESCRIP\_RD\_BL1\_MASK**)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_BL2**(n) (((uint32\_t)(n)&**ENET\_QOS\_TXDESCRIP\_RD\_BL1\_MASK**) << 16)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_TTSE\_MASK** (1UL << 30UL)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_IOC\_MASK** (1UL << 31UL)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_FL\_MASK** (0x7FFFUL)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_FL**(n) (((uint32\_t)(n)&**ENET\_QOS\_TXDESCRIP\_RD\_FL\_MASK**)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_CIC**(n) (((uint32\_t)(n)&0x3U) << 16U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_TSE\_MASK** (1UL << 18U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_SLOT**(n) (((uint32\_t)(n)&0xfU) << 19U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_SAIC**(n) (((uint32\_t)(n)&0x7U) << 23U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_CPC**(n) (((uint32\_t)(n)&0x3U) << 26U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_LDFD**(n) (((uint32\_t)(n)&0x3U) << 28U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_LD\_MASK** (1UL << 28U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_FD\_MASK** (1UL << 29U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_CTXT\_MASK** (1UL << 30U)
- #define **ENET\_QOS\_TXDESCRIP\_RD\_OWN\_MASK** (1UL << 31U)
- #define **ENET\_QOS\_TXDESCRIP\_WB\_TTSS\_MASK** (1UL << 17U)

*Defines for write back format.*

### Bit mask for interrupt enable type.

- #define **ENET\_QOS\_ABNORM\_INT\_MASK**
- #define **ENET\_QOS\_NORM\_INT\_MASK**

### Defines some Ethernet parameters.

- #define **ENET\_QOS\_RING\_NUM\_MAX** (5U)  
*The Maximum number of tx/rx descriptor rings.*
- #define **ENET\_QOS\_FRAME\_MAX\_FRAMELEN** (1518U)  
*Default maximum Ethernet frame size.*
- #define **ENET\_QOS\_FCS\_LEN** (4U)  
*Ethernet FCS length.*
- #define **ENET\_QOS\_ADDR\_ALIGNMENT** (0x3U)  
*Recommended Ethernet buffer alignment.*
- #define **ENET\_QOS\_BUFF\_ALIGNMENT** (8U)  
*Receive buffer alignment shall be 4bytes-aligned.*
- #define **ENET\_QOS\_MTL\_RXFIFOSIZE** (8192U)  
*The rx fifo size.*
- #define **ENET\_QOS\_MTL\_TXFIFOSIZE** (8192U)  
*The tx fifo size.*
- #define **ENET\_QOS\_MACINT\_ENUM\_OFFSET** (16U)  
*The offset for mac interrupt in enum type.*
- #define **ENET\_QOS\_RXP\_ENTRY\_COUNT** (256U)  
*RXP table entry count, implied by FRPES in MAC\_HW\_FEATURE3.*
- #define **ENET\_QOS\_RXP\_BUFFER\_SIZE** (256U)  
*RXP Buffer size, implied by FRPBS in MAC\_HW\_FEATURE3.*
- #define **ENET\_QOS\_EST\_WID** (24U)  
*Width of the time interval in Gate Control List.*
- #define **ENET\_QOS\_EST\_DEP** (512U)

Maximum depth of Gate Control List.

## Initialization and De-initialization

- void [ENET\\_QOS\\_GetDefaultConfig](#) ([enet\\_qos\\_config\\_t](#) \*config)  
*Gets the ENET default configuration structure.*
- [status\\_t](#) [ENET\\_QOS\\_Up](#) ([ENET\\_QOS\\_Type](#) \*base, const [enet\\_qos\\_config\\_t](#) \*config, [uint8\\_t](#) \*macAddr, [uint8\\_t](#) macCount, [uint32\\_t](#) refclkSrc\_Hz)  
*Initializes the ENET module.*
- [status\\_t](#) [ENET\\_QOS\\_Init](#) ([ENET\\_QOS\\_Type](#) \*base, const [enet\\_qos\\_config\\_t](#) \*config, [uint8\\_t](#) \*macAddr, [uint8\\_t](#) macCount, [uint32\\_t](#) refclkSrc\_Hz)  
*Initializes the ENET module.*
- void [ENET\\_QOS\\_Down](#) ([ENET\\_QOS\\_Type](#) \*base)  
*Stops the ENET module.*
- void [ENET\\_QOS\\_Deinit](#) ([ENET\\_QOS\\_Type](#) \*base)  
*Deinitializes the ENET module.*
- [uint32\\_t](#) [ENET\\_QOS\\_GetInstance](#) ([ENET\\_QOS\\_Type](#) \*base)  
*Get the ENET instance from peripheral base address.*
- [status\\_t](#) [ENET\\_QOS\\_DescriptorInit](#) ([ENET\\_QOS\\_Type](#) \*base, [enet\\_qos\\_config\\_t](#) \*config, [enet\\_qos\\_buffer\\_config\\_t](#) \*bufferConfig)  
*Initialize for all ENET descriptors.*
- [status\\_t](#) [ENET\\_QOS\\_RxBufferAllocAll](#) ([ENET\\_QOS\\_Type](#) \*base, [enet\\_qos\\_handle\\_t](#) \*handle)  
*Allocates Rx buffers for all BDs.*
- void [ENET\\_QOS\\_RxBufferFreeAll](#) ([ENET\\_QOS\\_Type](#) \*base, [enet\\_qos\\_handle\\_t](#) \*handle)  
*Frees Rx buffers in all BDs.*
- void [ENET\\_QOS\\_StartRxTx](#) ([ENET\\_QOS\\_Type](#) \*base, [uint8\\_t](#) txRingNum, [uint8\\_t](#) rxRingNum)  
*Starts the ENET rx/tx.*

## MII interface operation

- static void [ENET\\_QOS\\_SetMII](#) ([ENET\\_QOS\\_Type](#) \*base, [enet\\_qos\\_mii\\_speed\\_t](#) speed, [enet\\_qos\\_mii\\_duplex\\_t](#) duplex)  
*Sets the ENET MII speed and duplex.*
- void [ENET\\_QOS\\_SetSMI](#) ([ENET\\_QOS\\_Type](#) \*base, [uint32\\_t](#) csrClock\_Hz)  
*Sets the ENET SMI(serial management interface)- MII management interface.*
- static bool [ENET\\_QOS\\_IsSMIBusy](#) ([ENET\\_QOS\\_Type](#) \*base)  
*Checks if the SMI is busy.*
- static [uint16\\_t](#) [ENET\\_QOS\\_ReadSMIData](#) ([ENET\\_QOS\\_Type](#) \*base)  
*Reads data from the PHY register through SMI interface.*
- void [ENET\\_QOS\\_StartSMIRead](#) ([ENET\\_QOS\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) phyReg)  
*Starts an SMI read command.*
- void [ENET\\_QOS\\_StartSMIWrite](#) ([ENET\\_QOS\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) phyReg, [uint32\\_t](#) data)  
*Starts a SMI write command.*
- void [ENET\\_QOS\\_StartExtC45SMIWrite](#) ([ENET\\_QOS\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) device, [uint32\\_t](#) phyReg, [uint32\\_t](#) data)  
*Starts a SMI write command.*
- void [ENET\\_QOS\\_StartExtC45SMIRead](#) ([ENET\\_QOS\\_Type](#) \*base, [uint32\\_t](#) phyAddr, [uint32\\_t](#) device, [uint32\\_t](#) phyReg)  
*Starts a SMI read command.*



## Other basic operation

- static void [ENET\\_QOS\\_SetMacAddr](#) (ENET\_QOS\_Type \*base, uint8\_t \*macAddr, uint8\_t index)  
*Sets the ENET module Mac address.*
- void [ENET\\_QOS\\_GetMacAddr](#) (ENET\_QOS\_Type \*base, uint8\_t \*macAddr, uint8\_t index)  
*Gets the ENET module Mac address.*
- void [ENET\\_QOS\\_AddMulticastGroup](#) (ENET\_QOS\_Type \*base, uint8\_t \*address)  
*Adds the ENET\_QOS device to a multicast group.*
- void [ENET\\_QOS\\_LeaveMulticastGroup](#) (ENET\_QOS\_Type \*base, uint8\_t \*address)  
*Moves the ENET\_QOS device from a multicast group.*
- static void [ENET\\_QOS\\_AcceptAllMulticast](#) (ENET\_QOS\_Type \*base)  
*Enable ENET device to accept all multicast frames.*
- static void [ENET\\_QOS\\_RejectAllMulticast](#) (ENET\_QOS\_Type \*base)  
*ENET device reject to accept all multicast frames.*
- void [ENET\\_QOS\\_EnterPowerDown](#) (ENET\_QOS\_Type \*base, uint32\_t \*wakeFilter)  
*Set the MAC to enter into power down mode.*
- static void [ENET\\_QOS\\_ExitPowerDown](#) (ENET\_QOS\_Type \*base)  
*Set the MAC to exit power down mode.*
- [status\\_t](#) [ENET\\_QOS\\_EnableRxParser](#) (ENET\_QOS\_Type \*base, bool enable)  
*Enable/Disable Rx parser, please notice that for enable/disable Rx Parser, should better disable Receive first.*

## Interrupts.

- void [ENET\\_QOS\\_EnableInterrupts](#) (ENET\_QOS\_Type \*base, uint32\_t mask)  
*Enables the ENET DMA and MAC interrupts.*
- void [ENET\\_QOS\\_DisableInterrupts](#) (ENET\_QOS\_Type \*base, uint32\_t mask)  
*Disables the ENET DMA and MAC interrupts.*
- static uint32\_t [ENET\\_QOS\\_GetDmaInterruptStatus](#) (ENET\_QOS\_Type \*base, uint8\_t channel)  
*Gets the ENET DMA interrupt status flag.*
- static void [ENET\\_QOS\\_ClearDmaInterruptStatus](#) (ENET\_QOS\_Type \*base, uint8\_t channel, uint32\_t mask)  
*Clear the ENET DMA interrupt status flag.*
- static uint32\_t [ENET\\_QOS\\_GetMacInterruptStatus](#) (ENET\_QOS\_Type \*base)  
*Gets the ENET MAC interrupt status flag.*
- void [ENET\\_QOS\\_ClearMacInterruptStatus](#) (ENET\_QOS\_Type \*base, uint32\_t mask)  
*Clears the ENET mac interrupt events status flag.*

## Functional operation.

- static bool [ENET\\_QOS\\_IsTxDescriptorDmaOwn](#) (enet\_qos\_tx\_bd\_struct\_t \*txDesc)  
*Get the tx descriptor DMA Own flag.*
- void [ENET\\_QOS\\_SetupTxDescriptor](#) (enet\_qos\_tx\_bd\_struct\_t \*txDesc, void \*buffer1, uint32\_t bytes1, void \*buffer2, uint32\_t bytes2, uint32\_t framelen, bool intEnable, bool tsEnable, [enet\\_qos\\_desc\\_flag](#) flag, uint8\_t slotNum)  
*Setup a given tx descriptor.*
- static void [ENET\\_QOS\\_UpdateTxDescriptorTail](#) (ENET\_QOS\_Type \*base, uint8\_t channel, uint32\_t txDescTailAddrAlign)  
*Update the tx descriptor tail pointer.*
- static void [ENET\\_QOS\\_UpdateRxDescriptorTail](#) (ENET\_QOS\_Type \*base, uint8\_t channel, uint32\_t rxDescTailAddrAlign)

- *Update the rx descriptor tail pointer.*  
static uint32\_t [ENET\\_QOS\\_GetRxDescriptor](#) (enet\_qos\_rx\_bd\_struct\_t \*rxDesc)
- *Gets the context in the ENET rx descriptor.*  
void [ENET\\_QOS\\_UpdateRxDescriptor](#) (enet\_qos\_rx\_bd\_struct\_t \*rxDesc, void \*buffer1, void \*buffer2, bool intEnable, bool doubleBuffEnable)
- *Updates the buffers and the own status for a given rx descriptor.*  
status\_t [ENET\\_QOS\\_ConfigureRxParser](#) (ENET\_QOS\_Type \*base, enet\_qos\_rxp\_config\_t \*rxpConfig, uint16\_t entryCount)
- *Configure flexible rx parser.*  
status\_t [ENET\\_QOS\\_ReadRxParser](#) (ENET\_QOS\_Type \*base, enet\_qos\_rxp\_config\_t \*rxpConfig, uint16\_t entryIndex)
- *Read flexible rx parser configuration at specified index.*  
status\_t [ENET\\_QOS\\_EstProgramGcl](#) (ENET\_QOS\_Type \*base, enet\_qos\_est\_gcl\_t \*gcl, uint32\_t ptpClk\_Hz)
- *Program Gate Control List.*  
status\_t [ENET\\_QOS\\_EstReadGcl](#) (ENET\_QOS\_Type \*base, enet\_qos\_est\_gcl\_t \*gcl, uint32\_t listLen, bool hwList)
- *Read Gate Control List.*  
static void [ENET\\_QOS\\_FpeEnable](#) (ENET\_QOS\_Type \*base)
- *Enable Frame Preemption.*  
static void [ENET\\_QOS\\_FpeDisable](#) (ENET\_QOS\_Type \*base)
- *Disable Frame Preemption.*  
static void [ENET\\_QOS\\_FpeConfigPreemptable](#) (ENET\_QOS\_Type \*base, uint8\_t queueMask)
- *Configure preemptable transmit queues.*  
void [ENET\\_QOS\\_AVBConfigure](#) (ENET\_QOS\_Type \*base, const enet\_qos\_cbs\_config\_t \*config, uint8\_t queueIndex)
- *Sets the ENET AVB feature.*  
void [ENET\\_QOS\\_GetStatistics](#) (ENET\_QOS\_Type \*base, enet\_qos\_transfer\_stats\_t \*statistics)
- *Gets statistical data in transfer.*

## Transactional operation

- void [ENET\\_QOS\\_CreateHandler](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, enet\_qos\_config\_t \*config, enet\_qos\_buffer\_config\_t \*bufferConfig, enet\_qos\_callback\_t callback, void \*userData)
- *Create ENET Handler.*  
status\_t [ENET\\_QOS\\_GetRxFrameSize](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, uint32\_t \*length, uint8\_t channel)
- *Gets the size of the read frame.*  
status\_t [ENET\\_QOS\\_ReadFrame](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, uint8\_t \*data, uint32\_t length, uint8\_t channel, enet\_qos\_ptp\_time\_t \*ts)
- *Reads a frame from the ENET device.*  
status\_t [ENET\\_QOS\\_SendFrame](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, uint8\_t \*data, uint32\_t length, uint8\_t channel, bool isNeedTs, void \*context)
- *Transmits an ENET frame.*  
void [ENET\\_QOS\\_ReclaimTxDescriptor](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, uint8\_t channel)
- *Reclaim tx descriptors.*  
void [ENET\\_QOS\\_CommonIRQHandler](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle)
- *The ENET IRQ handler.*



- void [ENET\\_QOS\\_SetISRHandler](#) (ENET\_QOS\_Type \*base, enet\_qos\_isr\_t ISRHandler)  
*Set the second level IRQ handler, allow user to overwrite the default second level weak IRQ handler.*

## ENET Enhanced function operation

- [status\\_t ENET\\_QOS\\_Ptp1588CorrectTimerInCoarse](#) (ENET\_QOS\_Type \*base, [enet\\_qos\\_systime-\\_op](#) operation, uint32\_t second, uint32\_t nanosecond)  
*Correct the ENET PTP 1588 timer in coarse method.*
- [status\\_t ENET\\_QOS\\_Ptp1588CorrectTimerInFine](#) (ENET\_QOS\_Type \*base, uint32\_t addend)  
*Correct the ENET PTP 1588 timer in fine method.*
- static uint32\_t [ENET\\_QOS\\_Ptp1588GetAddend](#) (ENET\_QOS\_Type \*base)  
*Get the ENET Time stamp current addend value.*
- void [ENET\\_QOS\\_Ptp1588GetTimerNoIRQDisable](#) (ENET\_QOS\_Type \*base, uint64\_t \*second, uint32\_t \*nanosecond)  
*Gets the current ENET time from the PTP 1588 timer without IRQ disable.*
- static [status\\_t ENET\\_Ptp1588PpsControl](#) (ENET\_QOS\_Type \*base, [enet\\_qos\\_ptp\\_pps\\_instance\\_t](#) instance, [enet\\_qos\\_ptp\\_pps\\_trgt\\_mode\\_t](#) trgtMode, [enet\\_qos\\_ptp\\_pps\\_cmd\\_t](#) cmd)  
*Sets the ENET PTP 1588 PPS control.*
- [status\\_t ENET\\_QOS\\_Ptp1588PpsSetTrgtTime](#) (ENET\_QOS\_Type \*base, [enet\\_qos\\_ptp\\_pps\\_instance\\_t](#) instance, uint32\_t seconds, uint32\_t nanoseconds)  
*Sets the ENET OQS PTP 1588 PPS target time registers.*
- static void [ENET\\_QOS\\_Ptp1588PpsSetWidth](#) (ENET\_QOS\_Type \*base, [enet\\_qos\\_ptp\\_pps\\_instance\\_t](#) instance, uint32\_t width)  
*Sets the ENET OQS PTP 1588 PPS output signal interval.*
- static void [ENET\\_QOS\\_Ptp1588PpsSetInterval](#) (ENET\_QOS\_Type \*base, [enet\\_qos\\_ptp\\_pps\\_instance\\_t](#) instance, uint32\_t interval)  
*Sets the ENET OQS PTP 1588 PPS output signal width.*
- void [ENET\\_QOS\\_Ptp1588GetTimer](#) (ENET\_QOS\_Type \*base, uint64\_t \*second, uint32\_t \*nanosecond)  
*Gets the current ENET time from the PTP 1588 timer.*
- void [ENET\\_QOS\\_GetTxFrame](#) (enet\_qos\_handle\_t \*handle, [enet\\_qos\\_frame\\_info\\_t](#) \*txFrame, uint8\_t channel)  
*Gets the time stamp of the transmit frame.*
- [status\\_t ENET\\_QOS\\_GetRxFrame](#) (ENET\_QOS\_Type \*base, enet\_qos\_handle\_t \*handle, [enet\\_qos\\_rx\\_frame\\_struct\\_t](#) \*rxFrame, uint8\_t channel)  
*Receives one frame in specified BD ring with zero copy.*

## 81.2 Data Structure Documentation

### 81.2.1 struct enet\_qos\_rx\_bd\_struct\_t

They both has the same size with different region definition. so we define the read-format region as the receive descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the receive data process.

## Data Fields

- \_\_IO uint32\_t [buff1Addr](#)

- `__IO uint32_t reserved`  
*Buffer 1 address.*
- `__IO uint32_t buff2Addr`  
*Reserved.*
- `__IO uint32_t control`  
*Buffer 2 or next descriptor address.*
- `__IO uint32_t control`  
*Buffer 1/2 byte counts and control.*

### 81.2.2 struct enet\_qos\_tx\_bd\_struct\_t

They both has the same size with different region definition. so we define the read-format region as the transmit descriptor structure Use the read-format region mask bits in the descriptor initialization Use the write-back format region mask bits in the transmit data process.

#### Data Fields

- `__IO uint32_t buff1Addr`  
*Buffer 1 address.*
- `__IO uint32_t buff2Addr`  
*Buffer 2 address.*
- `__IO uint32_t buffLen`  
*Buffer 1/2 byte counts.*
- `__IO uint32_t controlStat`  
*TDES control and status word.*

### 81.2.3 struct enet\_qos\_ptp\_time\_t

#### Data Fields

- `uint64_t second`  
*Second.*
- `uint32_t nanosecond`  
*Nanosecond.*

#### Field Documentation

(1) `uint64_t enet_qos_ptp_time_t::second`

(2) `uint32_t enet_qos_ptp_time_t::nanosecond`

### 81.2.4 struct enet\_qos\_frame\_info\_t

#### Data Fields

- `void * context`

- *User specified data, could be buffer address for free.*
- bool [isTsAvail](#)  
*Flag indicates timestamp available status.*
- [enet\\_qos\\_ptp\\_time\\_t](#) [timeStamp](#)  
*Timestamp of frame.*

### 81.2.5 struct enet\_qos\_tx\_dirty\_ring\_t

#### Data Fields

- [enet\\_qos\\_frame\\_info\\_t](#) \* [txDirtyBase](#)  
*Dirty buffer descriptor base address pointer.*
- uint16\_t [txGenIdx](#)  
*tx generate index.*
- uint16\_t [txConsumIdx](#)  
*tx consume index.*
- uint16\_t [txRingLen](#)  
*tx ring length.*
- bool [isFull](#)  
*tx ring is full flag, add this parameter to avoid waste one element.*

#### Field Documentation

- (1) [enet\\_qos\\_frame\\_info\\_t](#)\* [enet\\_qos\\_tx\\_dirty\\_ring\\_t::txDirtyBase](#)
- (2) [uint16\\_t](#) [enet\\_qos\\_tx\\_dirty\\_ring\\_t::txGenIdx](#)
- (3) [uint16\\_t](#) [enet\\_qos\\_tx\\_dirty\\_ring\\_t::txConsumIdx](#)
- (4) [uint16\\_t](#) [enet\\_qos\\_tx\\_dirty\\_ring\\_t::txRingLen](#)
- (5) [bool](#) [enet\\_qos\\_tx\\_dirty\\_ring\\_t::isFull](#)

### 81.2.6 struct enet\_qos\_ptp\_config\_t

#### Data Fields

- bool [fineUpdateEnable](#)  
*Use the fine update.*
- uint32\_t [defaultAddend](#)  
*Default addend value when fine update is enable, could be  $2^{32} / (refClk\_Hz / ENET\_QOS\_MICRSECS\_ONESECOND / ENET\_QOS\_SYSTIME\_REQUIRED\_CLK\_MHZ)$ .*
- bool [ptp1588V2Enable](#)  
*The desired system time frequency.*
- [enet\\_qos\\_ts\\_rollover\\_type](#) [tsRollover](#)  
*1588 time nanosecond rollover.*

#### Field Documentation

(1) `bool enet_qos_ptp_config_t::fineUpdateEnable`

(2) `uint32_t enet_qos_ptp_config_t::defaultAddend`

(3) `bool enet_qos_ptp_config_t::ptp1588V2Enable`

Must be lower than reference clock. (Only used with fine correction method). ptp 1588 version 2 is used.

(4) `enet_qos_ts_rollover_type enet_qos_ptp_config_t::tsRollover`

### 81.2.7 struct enet\_qos\_est\_gate\_op\_t

### 81.2.8 struct enet\_qos\_est\_gcl\_t

#### Data Fields

- `bool enable`  
*Enable or disable EST.*
- `uint64_t cycleTime`  
*Base Time 32 bits seconds 32 bits nanoseconds.*
- `uint32_t extTime`  
*Cycle Time 32 bits seconds 32 bits nanoseconds.*
- `uint32_t numEntries`  
*Time Extension 32 bits seconds 32 bits nanoseconds.*
- `enet_qos_est_gate_op_t * opList`  
*Number of entries.*

### 81.2.9 struct enet\_qos\_rxp\_config\_t

#### Data Fields

- `uint32_t matchEnable`  
*4-byte match data used for comparing with incoming packet*
- `uint8_t acceptFrame: 1`  
*When matchEnable is set to 1, the matchData is used for comparing.*
- `uint8_t rejectFrame: 1`  
*When acceptFrame = 1 and data is matched, the frame will be sent to DMA channel.*
- `uint8_t inverseMatch: 1`  
*When rejectFrame = 1 and data is matched, the frame will be dropped.*
- `uint8_t nextControl: 1`  
*Inverse match.*
- `uint8_t reserved: 4`  
*Next instruction indexing control.*
- `uint8_t frameOffset`  
*Reserved control fields.*
- `uint8_t okIndex`  
*Frame offset in the packet data to be compared for match, in terms of 4 bytes.*

- uint8\_t [dmaChannel](#)  
*Memory Index to be used next.*
- uint32\_t [reserved2](#)  
*The DMA channel `enet_qos_rxp_dma_chn_t` used for receiving the frame when frame match and accept-Frame = 1.*

## Field Documentation

(1) uint8\_t `enet_qos_rxp_config_t::okIndex`

(2) uint8\_t `enet_qos_rxp_config_t::dmaChannel`

### 81.2.10 struct `enet_qos_buffer_config_t`

Note

1. The receive and transmit descriptor start address pointer and tail pointer must be word-aligned.
2. The recommended minimum tx/rx ring length is 4.
3. The tx/rx descriptor tail address shall be the address pointer to the address just after the end of the last last descriptor. because only the descriptors between the start address and the tail address will be used by DMA.
4. The descriptor address is the start address of all used contiguous memory. for example, the `rxDescStartAddrAlign` is the start address of `rxRingLen` contiguous descriptor memories for rx descriptor ring 0.
5. The "`*rxBufferstartAddr`" is the first element of `rxRingLen` ( $2 \times \text{rxRingLen}$  for double buffers) rx buffers. It means the `*rxBufferStartAddr` is the rx buffer for the first descriptor the `*rx-BufferStartAddr + 1` is the rx buffer for the second descriptor or the rx buffer for the second buffer in the first descriptor. so please make sure the `rxBufferStartAddr` is the address of a `rxRingLen` or  $2 \times \text{rxRingLen}$  array.

## Data Fields

- uint8\_t [rxRingLen](#)  
*The length of receive buffer descriptor ring.*
- uint8\_t [txRingLen](#)  
*The length of transmit buffer descriptor ring.*
- [enet\\_qos\\_tx\\_bd\\_struct\\_t](#) \* [txDescStartAddrAlign](#)  
*Aligned transmit descriptor start address.*
- [enet\\_qos\\_tx\\_bd\\_struct\\_t](#) \* [txDescTailAddrAlign](#)  
*Aligned transmit descriptor tail address.*
- [enet\\_qos\\_frame\\_info\\_t](#) \* [txDirtyStartAddr](#)  
*Start address of the dirty tx frame information.*
- [enet\\_qos\\_rx\\_bd\\_struct\\_t](#) \* [rxDescStartAddrAlign](#)  
*Aligned receive descriptor start address.*
- [enet\\_qos\\_rx\\_bd\\_struct\\_t](#) \* [rxDescTailAddrAlign](#)  
*Aligned receive descriptor tail address.*
- uint32\_t \* [rxBufferStartAddr](#)  
*Start address of the rx buffers.*

- uint32\_t [rxBuffSizeAlign](#)  
*Aligned receive data buffer size.*
- bool [rxBuffNeedMaintain](#)  
*Whether receive data buffer need cache maintain.*

#### Field Documentation

- (1) uint8\_t enet\_qos\_buffer\_config\_t::rxRingLen
- (2) uint8\_t enet\_qos\_buffer\_config\_t::txRingLen
- (3) enet\_qos\_tx\_bd\_struct\_t\* enet\_qos\_buffer\_config\_t::txDescStartAddrAlign
- (4) enet\_qos\_tx\_bd\_struct\_t\* enet\_qos\_buffer\_config\_t::txDescTailAddrAlign
- (5) enet\_qos\_frame\_info\_t\* enet\_qos\_buffer\_config\_t::txDirtyStartAddr
- (6) enet\_qos\_rx\_bd\_struct\_t\* enet\_qos\_buffer\_config\_t::rxDescStartAddrAlign
- (7) enet\_qos\_rx\_bd\_struct\_t\* enet\_qos\_buffer\_config\_t::rxDescTailAddrAlign
- (8) uint32\_t\* enet\_qos\_buffer\_config\_t::rxBufferStartAddr
- (9) uint32\_t enet\_qos\_buffer\_config\_t::rxBuffSizeAlign
- (10) bool enet\_qos\_buffer\_config\_t::rxBuffNeedMaintain

### 81.2.11 struct enet\_qos\_cbs\_config\_t

#### Data Fields

- uint16\_t [sendSlope](#)  
*Send slope configuration.*
- uint16\_t [idleSlope](#)  
*Idle slope configuration.*
- uint32\_t [highCredit](#)  
*High credit.*
- uint32\_t [lowCredit](#)  
*Low credit.*

#### Field Documentation

- (1) uint16\_t enet\_qos\_cbs\_config\_t::sendSlope
- (2) uint16\_t enet\_qos\_cbs\_config\_t::idleSlope
- (3) uint32\_t enet\_qos\_cbs\_config\_t::highCredit
- (4) uint32\_t enet\_qos\_cbs\_config\_t::lowCredit

## 81.2.12 struct enet\_qos\_queue\_tx\_config\_t

### Data Fields

- [enet\\_qos\\_queue\\_mode\\_t mode](#)  
*tx queue mode configuration.*
- [uint32\\_t weight](#)  
*Refer to the MTL TxQ Quantum Weight register.*
- [uint32\\_t priority](#)  
*Refer to Transmit Queue Priority Mapping register.*
- [enet\\_qos\\_cbs\\_config\\_t \\* cbsConfig](#)  
*CBS configuration if queue use AVB mode.*

### Field Documentation

- (1) `enet_qos_queue_mode_t enet_qos_queue_tx_config_t::mode`
- (2) `uint32_t enet_qos_queue_tx_config_t::weight`
- (3) `uint32_t enet_qos_queue_tx_config_t::priority`
- (4) `enet_qos_cbs_config_t* enet_qos_queue_tx_config_t::cbsConfig`

## 81.2.13 struct enet\_qos\_queue\_rx\_config\_t

### Data Fields

- [enet\\_qos\\_queue\\_mode\\_t mode](#)  
*rx queue mode configuration.*
- [uint8\\_t mapChannel](#)  
*tx queue map dma channel.*
- [uint32\\_t priority](#)  
*Rx queue priority.*
- [enet\\_qos\\_rx\\_queue\\_route\\_t packetRoute](#)  
*Receive packet routing.*

### Field Documentation

- (1) `enet_qos_queue_mode_t enet_qos_queue_rx_config_t::mode`
- (2) `uint8_t enet_qos_queue_rx_config_t::mapChannel`
- (3) `uint32_t enet_qos_queue_rx_config_t::priority`
- (4) `enet_qos_rx_queue_route_t enet_qos_queue_rx_config_t::packetRoute`

### 81.2.14 struct enet\_qos\_multiqueue\_config\_t

#### Data Fields

- [enet\\_qos\\_dma\\_burstlen](#) `burstLen`  
*Burst len for the multi-queue.*
- `uint8_t` [txQueueUse](#)  
*Used Tx queue count.*
- [enet\\_qos\\_mtl\\_multiqueue\\_txsche](#) `mtltxSche`  
*Transmit schedule for multi-queue.*
- [enet\\_qos\\_queue\\_tx\\_config\\_t](#) `txQueueConfig` [ENET\_QOS\_RING\_NUM\_MAX]  
*Tx Queue configuration.*
- `uint8_t` [rxQueueUse](#)  
*Used Rx queue count.*
- [enet\\_qos\\_mtl\\_multiqueue\\_rxsche](#) `mtlrxSche`  
*Receive schedule for multi-queue.*
- [enet\\_qos\\_queue\\_rx\\_config\\_t](#) `rxQueueConfig` [ENET\_QOS\_RING\_NUM\_MAX]  
*Rx Queue configuration.*

#### Field Documentation

- (1) `enet_qos_dma_burstlen` `enet_qos_multiqueue_config_t::burstLen`
- (2) `uint8_t` `enet_qos_multiqueue_config_t::txQueueUse`
- (3) `enet_qos_mtl_multiqueue_txsche` `enet_qos_multiqueue_config_t::mtltxSche`
- (4) `enet_qos_queue_tx_config_t` `enet_qos_multiqueue_config_t::txQueueConfig`[ENET\_QOS\_RING\_NUM\_MAX]
- (5) `uint8_t` `enet_qos_multiqueue_config_t::rxQueueUse`
- (6) `enet_qos_mtl_multiqueue_rxsche` `enet_qos_multiqueue_config_t::mtlrxSche`
- (7) `enet_qos_queue_rx_config_t` `enet_qos_multiqueue_config_t::rxQueueConfig`[ENET\_QOS\_RING\_NUM\_MAX]

### 81.2.15 struct enet\_qos\_config\_t

#### Note

Default the signal queue is used so the "`*multiqueueCfg`" is set default with NULL. Set the pointer with a valid configuration pointer if the multiple queues are required. If multiple queue is enabled, please make sure the buffer configuration for all are prepared also.

#### Data Fields

- `uint16_t` [specialControl](#)



- *The logic or of `enet_qos_special_config_t`.*
- `enet_qos_multiqueue_config_t` \* `multiqueueCfg`  
*Use multi-queue.*
- `enet_qos_mii_mode_t` `miiMode`  
*MII mode.*
- `enet_qos_mii_speed_t` `miiSpeed`  
*MII Speed.*
- `enet_qos_mii_duplex_t` `miiDuplex`  
*MII duplex.*
- `uint16_t` `pauseDuration`  
*Used in the tx flow control frame, only valid when `kENET_QOS_FlowControlEnable` is set.*
- `enet_qos_ptp_config_t` \* `ptpConfig`  
*PTP 1588 feature configuration.*
- `uint32_t` `csrClock_Hz`  
*CSR clock frequency in HZ.*
- `enet_qos_rx_alloc_callback_t` `rxBuffAlloc`  
*Callback to alloc memory, must be provided for zero-copy Rx.*
- `enet_qos_rx_free_callback_t` `rxBuffFree`  
*Callback to free memory, must be provided for zero-copy Rx.*

### Field Documentation

- (1) `enet_qos_multiqueue_config_t`\* `enet_qos_config_t::multiqueueCfg`
- (2) `enet_qos_mii_mode_t` `enet_qos_config_t::miiMode`
- (3) `enet_qos_mii_speed_t` `enet_qos_config_t::miiSpeed`
- (4) `enet_qos_mii_duplex_t` `enet_qos_config_t::miiDuplex`
- (5) `uint16_t` `enet_qos_config_t::pauseDuration`
- (6) `uint32_t` `enet_qos_config_t::csrClock_Hz`
- (7) `enet_qos_rx_alloc_callback_t` `enet_qos_config_t::rxBuffAlloc`
- (8) `enet_qos_rx_free_callback_t` `enet_qos_config_t::rxBuffFree`

### 81.2.16 struct `enet_qos_tx_bd_ring_t`

#### Data Fields

- `enet_qos_tx_bd_struct_t` \* `txBdBase`  
*Buffer descriptor base address pointer.*
- `uint16_t` `txGenIdx`  
*tx generate index.*
- `uint16_t` `txConsumIdx`  
*tx consume index.*
- `volatile uint16_t` `txDescUsed`  
*tx descriptor used number.*

- `uint16_t txRingLen`  
*tx ring length.*

#### Field Documentation

- (1) `enet_qos_tx_bd_struct_t* enet_qos_tx_bd_ring_t::txBdBase`
- (2) `uint16_t enet_qos_tx_bd_ring_t::txGenIdx`
- (3) `uint16_t enet_qos_tx_bd_ring_t::txConsumeIdx`
- (4) `volatile uint16_t enet_qos_tx_bd_ring_t::txDescUsed`
- (5) `uint16_t enet_qos_tx_bd_ring_t::txRingLen`

### 81.2.17 struct enet\_qos\_rx\_bd\_ring\_t

#### Data Fields

- `enet_qos_rx_bd_struct_t * rxBdBase`  
*Buffer descriptor base address pointer.*
- `uint16_t rxGenIdx`  
*The current available receive buffer descriptor pointer.*
- `uint16_t rxRingLen`  
*Receive ring length.*
- `uint32_t rxBuffSizeAlign`  
*Receive buffer size.*

#### Field Documentation

- (1) `enet_qos_rx_bd_struct_t* enet_qos_rx_bd_ring_t::rxBdBase`
- (2) `uint16_t enet_qos_rx_bd_ring_t::rxGenIdx`
- (3) `uint16_t enet_qos_rx_bd_ring_t::rxRingLen`
- (4) `uint32_t enet_qos_rx_bd_ring_t::rxBuffSizeAlign`

### 81.2.18 struct \_enet\_qos\_handle

#### Data Fields

- `uint8_t txQueueUse`  
*Used tx queue count.*
- `uint8_t rxQueueUse`  
*Used rx queue count.*
- `bool doubleBuffEnable`  
*The double buffer is used in the descriptor.*
- `bool rxintEnable`

- *Rx interrupt enabled.*  
• bool `rxMaintainEnable` [ENET\_QOS\_RING\_NUM\_MAX]
- *Rx buffer cache maintain enabled.*  
• `enet_qos_rx_bd_ring_t` `rxBdRing` [ENET\_QOS\_RING\_NUM\_MAX]
- *Receive buffer descriptor.*  
• `enet_qos_tx_bd_ring_t` `txBdRing` [ENET\_QOS\_RING\_NUM\_MAX]
- *Transmit buffer descriptor.*  
• `enet_qos_tx_dirty_ring_t` `txDirtyRing` [ENET\_QOS\_RING\_NUM\_MAX]
- *Transmit dirty buffers addresses.*  
• `uint32_t` \* `rxBufferStartAddr` [ENET\_QOS\_RING\_NUM\_MAX]
- *Rx buffer start address for reInitialize.*  
• `enet_qos_callback_t` `callback`
- *Callback function.*  
• void \* `userData`
- *Callback function parameter.*  
• `uint8_t` `multicastCount` [64]
- *Multicast collisions counter.*  
• `enet_qos_rx_alloc_callback_t` `rxBuffAlloc`
- *Callback to alloc memory, must be provided for zero-copy Rx.*  
• `enet_qos_rx_free_callback_t` `rxBuffFree`
- *Callback to free memory, must be provided for zero-copy Rx.*

## Field Documentation

- (1) `uint8_t` `enet_qos_handle_t::txQueueUse`
- (2) `uint8_t` `enet_qos_handle_t::rxQueueUse`
- (3) `bool` `enet_qos_handle_t::doubleBuffEnable`
- (4) `bool` `enet_qos_handle_t::rxintEnable`
- (5) `bool` `enet_qos_handle_t::rxMaintainEnable`[ENET\_QOS\_RING\_NUM\_MAX]
- (6) `enet_qos_rx_bd_ring_t` `enet_qos_handle_t::rxBdRing`[ENET\_QOS\_RING\_NUM\_MAX]
- (7) `enet_qos_tx_bd_ring_t` `enet_qos_handle_t::txBdRing`[ENET\_QOS\_RING\_NUM\_MAX]
- (8) `enet_qos_tx_dirty_ring_t` `enet_qos_handle_t::txDirtyRing`[ENET\_QOS\_RING\_NUM\_MAX]
- (9) `uint32_t`\* `enet_qos_handle_t::rxBufferStartAddr`[ENET\_QOS\_RING\_NUM\_MAX]
- (10) `enet_qos_callback_t` `enet_qos_handle_t::callback`
- (11) `void*` `enet_qos_handle_t::userData`
- (12) `enet_qos_rx_alloc_callback_t` `enet_qos_handle_t::rxBuffAlloc`
- (13) `enet_qos_rx_free_callback_t` `enet_qos_handle_t::rxBuffFree`

## 81.2.19 struct enet\_qos\_buffer\_struct\_t

### Data Fields

- void \* [buffer](#)  
*The buffer store the whole or partial frame.*
- uint16\_t [length](#)  
*The byte length of this buffer.*

### Field Documentation

(1) void\* enet\_qos\_buffer\_struct\_t::buffer

(2) uint16\_t enet\_qos\_buffer\_struct\_t::length

## 81.2.20 struct enet\_qos\_rx\_frame\_error\_t

### Data Fields

- bool [rxDstAddrFilterErr](#): 1  
*Destination Address Filter Fail.*
- bool [rxSrcAddrFilterErr](#): 1  
*SA Address Filter Fail.*
- bool [rxDribbleErr](#): 1  
*Dribble error.*
- bool [rxReceiveErr](#): 1  
*Receive error.*
- bool [rxOverflowErr](#): 1  
*Receive over flow.*
- bool [rxWatchDogErr](#): 1  
*Watch dog timeout.*
- bool [rxGaintPacketErr](#): 1  
*Receive gaint packet.*
- bool [rxCrcErr](#): 1  
*Receive CRC error.*

### Field Documentation

(1) bool enet\_qos\_rx\_frame\_error\_t::rxDstAddrFilterErr

(2) bool enet\_qos\_rx\_frame\_error\_t::rxSrcAddrFilterErr

(3) bool enet\_qos\_rx\_frame\_error\_t::rxDribbleErr

(4) bool enet\_qos\_rx\_frame\_error\_t::rxReceiveErr

(5) bool enet\_qos\_rx\_frame\_error\_t::rxOverflowErr

(6) bool enet\_qos\_rx\_frame\_error\_t::rxWatchDogErr

(7) `bool enet_qos_rx_frame_error_t::rxGaintPacketErr`

(8) `bool enet_qos_rx_frame_error_t::rxCrcErr`

## 81.2.21 `struct enet_qos_rx_frame_struct_t`

### Data Fields

- `enet_qos_buffer_struct_t * rxBuffArray`  
*Rx frame buffer structure.*
- `uint16_t totLen`  
*Rx frame total length.*
- `enet_qos_rx_frame_attribute_t rxAttribute`  
*Rx frame attribute structure.*
- `enet_qos_rx_frame_error_t rxFrameError`  
*Rx frame error.*

### Field Documentation

(1) `enet_qos_buffer_struct_t* enet_qos_rx_frame_struct_t::rxBuffArray`

(2) `uint16_t enet_qos_rx_frame_struct_t::totLen`

(3) `enet_qos_rx_frame_attribute_t enet_qos_rx_frame_struct_t::rxAttribute`

(4) `enet_qos_rx_frame_error_t enet_qos_rx_frame_struct_t::rxFrameError`

## 81.2.22 `struct enet_qos_transfer_stats_t`

### Data Fields

- `uint32_t statsRxFrameCount`  
*Rx frame number.*
- `uint32_t statsRxCrcErr`  
*Rx frame number with CRC error.*
- `uint32_t statsRxAlignErr`  
*Rx frame number with alignment error.*
- `uint32_t statsRxLengthErr`  
*Rx frame length field doesn't equal to packet size.*
- `uint32_t statsRxFifoOverflowErr`  
*Rx FIFO overflow count.*
- `uint32_t statsTxFrameCount`  
*Tx frame number.*
- `uint32_t statsTxFifoUnderRunErr`  
*Tx FIFO underrun count.*

### Field Documentation

(1) `uint32_t enet_qos_transfer_stats_t::statsRxFrameCount`

- (2) `uint32_t enet_qos_transfer_stats_t::statsRxCrcErr`
- (3) `uint32_t enet_qos_transfer_stats_t::statsRxAlignErr`
- (4) `uint32_t enet_qos_transfer_stats_t::statsRxLengthErr`
- (5) `uint32_t enet_qos_transfer_stats_t::statsRxFifoOverflowErr`
- (6) `uint32_t enet_qos_transfer_stats_t::statsTxFrameCount`
- (7) `uint32_t enet_qos_transfer_stats_t::statsTxFifoUnderRunErr`

### 81.3 Macro Definition Documentation

81.3.1 **#define FSL\_ENET\_QOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))**

81.3.2 **#define ENET\_QOS\_RXDESCRIP\_RD\_BUFF1VALID\_MASK (1UL << 24U)**

Buffer1 address valid.

81.3.3 **#define ENET\_QOS\_RXDESCRIP\_RD\_BUFF2VALID\_MASK (1UL << 25U)**

81.3.4 **#define ENET\_QOS\_RXDESCRIP\_RD\_IOC\_MASK (1UL << 30U)**

81.3.5 **#define ENET\_QOS\_RXDESCRIP\_RD\_OWN\_MASK (1UL << 31U)**

81.3.6 **#define ENET\_QOS\_RXDESCRIP\_WR\_ERR\_MASK ((1UL << 3U) | (1UL << 7U))**

81.3.7 **#define ENET\_QOS\_TXDESCRIP\_RD\_BL1\_MASK (0x3fffUL)**

81.3.8 **#define ENET\_QOS\_TXDESCRIP\_WB\_TTSS\_MASK (1UL << 17U)**

81.3.9 **#define ENET\_QOS\_RING\_NUM\_MAX (5U)**

81.3.10 **#define ENET\_QOS\_FRAME\_MAX\_FRAMELEN (1518U)**

81.3.11 **#define ENET\_QOS\_FCS\_LEN (4U)**

81.3.12 **#define ENET\_QOS\_ADDR\_ALIGNMENT (0x3U)**

**81.3.13** `#define ENET_QOS_BUFF_ALIGNMENT (8U)`

**81.3.14** `#define ENET_QOS_MTL_RXFIFOSIZE (8192U)`

**81.3.15** `#define ENET_QOS_MTL_TXFIFOSIZE (8192U)`

**81.3.16** `#define ENET_QOS_MACINT_ENUM_OFFSET (16U)`

## 81.4 Typedef Documentation

**81.4.1** `typedef void>(* enet_qos_rx_alloc_callback_t)(ENET_QOS_Type *base, void *userData, uint8_t channel)`

**81.4.2** `typedef void(* enet_qos_rx_free_callback_t)(ENET_QOS_Type *base, void *buffer, void *userData, uint8_t channel)`

**81.4.3** `typedef void(* enet_qos_callback_t)(ENET_QOS_Type *base, enet_qos_handle_t *handle, enet_qos_event_t event, uint8_t channel, void *userData)`

## 81.5 Enumeration Type Documentation

### 81.5.1 anonymous enum

#### Enumerator

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

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

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

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

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

*kStatus\_ENET\_QOS\_TxFrameBusy* Transmit descriptors are under process.

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

*kStatus\_ENET\_QOS\_TxFrameOverLen* Transmit oversize.

*kStatus\_ENET\_QOS\_Est\_SwListBusy* SW Gcl List not yet processed by HW.

*kStatus\_ENET\_QOS\_Est\_SwListWriteAbort* SW Gcl List write aborted .

*kStatus\_ENET\_QOS\_Est\_InvalidParameter* Invalid parameter in Gcl List .

*kStatus\_ENET\_QOS\_Est\_BtrError* Base Time Error when loading list.

*kStatus\_ENET\_QOS\_TrgtBusy* Target time register busy.

*kStatus\_ENET\_QOS\_Timeout* Target time register busy.

*kStatus\_ENET\_QOS\_PpsBusy* Pps command busy.

### 81.5.2 enum enet\_qos\_mii\_mode\_t

Enumerator

*kENET\_QOS\_MiiMode* MII mode for data interface.  
*kENET\_QOS\_RgmiiMode* RGMII mode for data interface.  
*kENET\_QOS\_RmiiMode* RMII mode for data interface.

### 81.5.3 enum enet\_qos\_mii\_speed\_t

Enumerator

*kENET\_QOS\_MiiSpeed10M* Speed 10 Mbps.  
*kENET\_QOS\_MiiSpeed100M* Speed 100 Mbps.  
*kENET\_QOS\_MiiSpeed1000M* Speed 1000 Mbps.  
*kENET\_QOS\_MiiSpeed2500M* Speed 2500 Mbps.

### 81.5.4 enum enet\_qos\_mii\_duplex\_t

Enumerator

*kENET\_QOS\_MiiHalfDuplex* Half duplex mode.  
*kENET\_QOS\_MiiFullDuplex* Full duplex mode.

### 81.5.5 enum enet\_qos\_mii\_normal\_opcode

Enumerator

*kENET\_QOS\_MiiWriteFrame* Write frame operation for a valid MII management frame.  
*kENET\_QOS\_MiiReadFrame* Read frame operation for a valid MII management frame.

### 81.5.6 enum enet\_qos\_dma\_burstlen

Enumerator

*kENET\_QOS\_BurstLen1* DMA burst length 1.  
*kENET\_QOS\_BurstLen2* DMA burst length 2.  
*kENET\_QOS\_BurstLen4* DMA burst length 4.  
*kENET\_QOS\_BurstLen8* DMA burst length 8.  
*kENET\_QOS\_BurstLen16* DMA burst length 16.



*kENET\_QOS\_BurstLen32* DMA burst length 32.  
*kENET\_QOS\_BurstLen64* DMA burst length 64. eight times enabled.  
*kENET\_QOS\_BurstLen128* DMA burst length 128. eight times enabled.  
*kENET\_QOS\_BurstLen256* DMA burst length 256. eight times enabled.

### 81.5.7 enum enet\_qos\_desc\_flag

Enumerator

*kENET\_QOS\_MiddleFlag* It's a middle descriptor of the frame.  
*kENET\_QOS\_LastFlagOnly* It's the last descriptor of the frame.  
*kENET\_QOS\_FirstFlagOnly* It's the first descriptor of the frame.  
*kENET\_QOS\_FirstLastFlag* It's the first and last descriptor of the frame.

### 81.5.8 enum enet\_qos\_systime\_op

Enumerator

*kENET\_QOS\_SystimeAdd* System time add to.  
*kENET\_QOS\_SystimeSubtract* System time subtract.

### 81.5.9 enum enet\_qos\_ts\_rollover\_type

Enumerator

*kENET\_QOS\_BinaryRollover* System time binary rollover.  
*kENET\_QOS\_DigitalRollover* System time digital rollover.

### 81.5.10 enum enet\_qos\_special\_config\_t

These control flags are provided for special user requirements. Normally, there is no need to set this control flags for ENET initialization. But if you have some special requirements, set the flags to specialControl in the [enet\\_qos\\_config\\_t](#).

Note

"kENET\_QOS\_StoreAndForward" is recommended to be set.

Enumerator

*kENET\_QOS\_DescDoubleBuffer* The double buffer is used in the tx/rx descriptor.

*kENET\_QOS\_StoreAndForward* The rx/tx store and forward enable.  
*kENET\_QOS\_PromiscuousEnable* The promiscuous enabled.  
*kENET\_QOS\_FlowControlEnable* The flow control enabled.  
*kENET\_QOS\_BroadCastRxDisable* The broadcast disabled.  
*kENET\_QOS\_MulticastAllEnable* All multicast are passed.  
*kENET\_QOS\_8023AS2KPacket* 8023as support for 2K packets.  
*kENET\_QOS\_HashMulticastEnable* The multicast packets are filtered through hash table.

### 81.5.11 enum enet\_qos\_dma\_interrupt\_enable\_t

This enumeration uses one-hot encoding to allow a logical OR of multiple members.

Enumerator

*kENET\_QOS\_DmaTx* Tx interrupt.  
*kENET\_QOS\_DmaTxStop* Tx stop interrupt.  
*kENET\_QOS\_DmaTxBuffUnavail* Tx buffer unavailable.  
*kENET\_QOS\_DmaRx* Rx interrupt.  
*kENET\_QOS\_DmaRxBuffUnavail* Rx buffer unavailable.  
*kENET\_QOS\_DmaRxStop* Rx stop.  
*kENET\_QOS\_DmaRxWatchdogTimeout* Rx watchdog timeout.  
*kENET\_QOS\_DmaEarlyTx* Early transmit.  
*kENET\_QOS\_DmaEarlyRx* Early receive.  
*kENET\_QOS\_DmaBusErr* Fatal bus error.

### 81.5.12 enum enet\_qos\_mac\_interrupt\_enable\_t

This enumeration uses one-hot encoding to allow a logical OR of multiple members.

### 81.5.13 enum enet\_qos\_event\_t

Enumerator

*kENET\_QOS\_RxIntEvent* Receive interrupt event.  
*kENET\_QOS\_TxIntEvent* Transmit interrupt event.  
*kENET\_QOS\_WakeUpIntEvent* Wake up interrupt event.  
*kENET\_QOS\_TimeStampIntEvent* Time stamp interrupt event.

**81.5.14 enum enet\_qos\_queue\_mode\_t**

Enumerator

*kENET\_QOS\_AVB\_Mode* Enable queue in AVB mode.*kENET\_QOS\_DCB\_Mode* Enable queue in DCB mode.**81.5.15 enum enet\_qos\_mtl\_multiqueue\_txsche**

Enumerator

*kENET\_QOS\_txWeightRR* Tx weight round-robin.*kENET\_QOS\_txWeightFQ* Tx weight fair queuing.*kENET\_QOS\_txDefictWeightRR* Tx deficit weighted round-robin.*kENET\_QOS\_txStrPrio* Tx strict priority.**81.5.16 enum enet\_qos\_mtl\_multiqueue\_rxsche**

Enumerator

*kENET\_QOS\_rxStrPrio* Rx strict priority, Queue 0 has the lowest priority.*kENET\_QOS\_rxWeightStrPrio* Weighted Strict Priority.**81.5.17 enum enet\_qos\_mtl\_rxqueuemap\_t**

Enumerator

*kENET\_QOS\_StaticDirctMap* The received fame in rx Qn(n = 0,1) directly map to dma channel n.*kENET\_QOS\_DynamicMap* The received frame in rx Qn(n = 0,1) map to the dma channel m(m = 0,1) related with the same Mac.**81.5.18 enum enet\_qos\_rx\_queue\_route\_t****81.5.19 enum enet\_qos\_ptp\_event\_type\_t**

Enumerator

*kENET\_QOS\_PtpEventMsgType* PTP event message type.*kENET\_QOS\_PtpSrcPortIdLen* PTP message sequence id length.*kENET\_QOS\_PtpEventPort* PTP event port number.*kENET\_QOS\_PtpGnrlPort* PTP general port number.

**81.5.20 enum enet\_qos\_ptp\_pps\_instance\_t**

Enumerator

*kENET\_QOS\_PtpPpsInstance0* PPS instance 0.  
*kENET\_QOS\_PtpPpsInstance1* PPS instance 1.  
*kENET\_QOS\_PtpPpsInstance2* PPS instance 2.  
*kENET\_QOS\_PtpPpsInstance3* PPS instance 3.

**81.5.21 enum enet\_qos\_ptp\_pps\_trgt\_mode\_t**

Enumerator

*kENET\_QOS\_PtpPpsTrgtModeOnlyInt* Only interrupts.  
*kENET\_QOS\_PtpPpsTrgtModeIntSt* Both interrupt and output signal.  
*kENET\_QOS\_PtpPpsTrgtModeOnlySt* Only output signal.

**81.5.22 enum enet\_qos\_ptp\_pps\_cmd\_t**

Enumerator

*kENET\_QOS\_PtpPpsCmdNC* No Command.  
*kENET\_QOS\_PtpPpsCmdSSP* Start Single Pulse.  
*kENET\_QOS\_PtpPpsCmdSPT* Start Pulse Train.  
*kENET\_QOS\_PtpPpsCmdCS* Cancel Start.  
*kENET\_QOS\_PtpPpsCmdSPTAT* Stop Pulse Train At Time.  
*kENET\_QOS\_PtpPpsCmdSPTI* Stop Pulse Train Immediately.  
*kENET\_QOS\_PtpPpsCmdCSPT* Cancel Stop Pulse Train.

**81.5.23 enum enet\_qos\_ets\_list\_length\_t**

Enumerator

*kENET\_QOS\_Ets\_List\_64* List length of 64.  
*kENET\_QOS\_Ets\_List\_128* List length of 128.  
*kENET\_QOS\_Ets\_List\_256* List length of 256.  
*kENET\_QOS\_Ets\_List\_512* List length of 512.  
*kENET\_QOS\_Ets\_List\_1024* List length of 1024.

### 81.5.24 enum enet\_qos\_ets\_gccr\_addr\_t

Enumerator

*kENET\_QOS\_Ets\_btr\_low* BTR Low.  
*kENET\_QOS\_Ets\_btr\_high* BTR High.  
*kENET\_QOS\_Ets\_ctr\_low* CTR Low.  
*kENET\_QOS\_Ets\_ctr\_high* CTR High.  
*kENET\_QOS\_Ets\_ter* TER.  
*kENET\_QOS\_Ets\_llr* LLR.

### 81.5.25 enum enet\_qos\_rxp\_dma\_chn\_t

Enumerator

*kENET\_QOS\_Rxp\_DMACHn0* DMA Channel 0 used for RXP entry match.  
*kENET\_QOS\_Rxp\_DMACHn1* DMA Channel 1 used for RXP entry match.  
*kENET\_QOS\_Rxp\_DMACHn2* DMA Channel 2 used for RXP entry match.  
*kENET\_QOS\_Rxp\_DMACHn3* DMA Channel 3 used for RXP entry match.  
*kENET\_QOS\_Rxp\_DMACHn4* DMA Channel 4 used for RXP entry match.

## 81.6 Function Documentation

### 81.6.1 void ENET\_QOS\_SetSYSControl ( enet\_qos\_mii\_mode\_t *miiMode* )

Note

User needs to provide the implementation because the implementation is SoC specific. This function set the phy selection and enable clock. It should be called before any other ethernet operation.

Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>miiMode</i> | The MII/RGMII/RMII mode for interface between the phy and Ethernet. |
|----------------|---------------------------------------------------------------------|

### 81.6.2 void ENET\_QOS\_GetDefaultConfig ( enet\_qos\_config\_t \* *config* )

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

```
enet_qos_config_t config;
ENET_QOS_GetDefaultConfig(&config);
```

## Parameters

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

### 81.6.3 **status\_t ENET\_QOS\_Up ( ENET\_QOS\_Type \* *base*, const enet\_qos\_config\_t \* *config*, uint8\_t \* *macAddr*, uint8\_t *macCount*, uint32\_t *refclkSrc\_Hz* )**

This function initializes it with the ENET basic configuration.

## Parameters

|                     |                                                                                                                                                                                                                               |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | ENET peripheral base address.                                                                                                                                                                                                 |
| <i>config</i>       | ENET mac configuration structure pointer. The "enet_qos_config_t" type mac configuration return from ENET_QOS_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods. |
| <i>macAddr</i>      | Pointer to ENET mac address array of Ethernet device. This MAC address should be provided.                                                                                                                                    |
| <i>macCount</i>     | Count of macAddr in the ENET mac address array                                                                                                                                                                                |
| <i>refclkSrc_Hz</i> | ENET input reference clock.                                                                                                                                                                                                   |

### 81.6.4 **status\_t ENET\_QOS\_Init ( ENET\_QOS\_Type \* *base*, const enet\_qos\_config\_t \* *config*, uint8\_t \* *macAddr*, uint8\_t *macCount*, uint32\_t *refclkSrc\_Hz* )**

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

## Parameters

|                |                                                                                                                                                                                                                               |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                                                                                                                                                 |
| <i>config</i>  | ENET mac configuration structure pointer. The "enet_qos_config_t" type mac configuration return from ENET_QOS_GetDefaultConfig can be used directly. It is also possible to verify the Mac configuration using other methods. |
| <i>macAddr</i> | Pointer to ENET mac address array of Ethernet device. This MAC address should be provided.                                                                                                                                    |

|                     |                                                |
|---------------------|------------------------------------------------|
| <i>macCount</i>     | Count of macAddr in the ENET mac address array |
| <i>refclkSrc_Hz</i> | ENET input reference clock.                    |

### 81.6.5 void ENET\_QOS\_Down ( ENET\_QOS\_Type \* *base* )

This function disables the ENET module.

Parameters

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

### 81.6.6 void ENET\_QOS\_Deinit ( ENET\_QOS\_Type \* *base* )

This function gates the module clock and disables the ENET module.

Parameters

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

### 81.6.7 uint32\_t ENET\_QOS\_GetInstance ( ENET\_QOS\_Type \* *base* )

Parameters

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

Returns

ENET instance.

### 81.6.8 status\_t ENET\_QOS\_DescriptorInit ( ENET\_QOS\_Type \* *base*, enet\_qos\_config\_t \* *config*, enet\_qos\_buffer\_config\_t \* *bufferConfig* )

Note

This function is do all tx/rx descriptors initialization. Because this API read all interrupt registers first and then set the interrupt flag for all descriptors, if the interrupt register is set. so the descriptor initialization should be called after [ENET\\_QOS\\_Init\(\)](#), [ENET\\_QOS\\_EnableInterrupts\(\)](#) and [ENET\\_QOS\\_CreateHandle\(\)](#)(if transactional APIs are used).

## Parameters

|                     |                               |
|---------------------|-------------------------------|
| <i>base</i>         | ENET peripheral base address. |
| <i>config</i>       | The configuration for ENET.   |
| <i>bufferConfig</i> | All buffers configuration.    |

### 81.6.9 status\_t ENET\_QOS\_RxBufferAllocAll ( ENET\_QOS\_Type \* *base*, enet\_qos\_handle\_t \* *handle* )

It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will populate initial buffers in all BDs for receiving. Then [ENET\\_QOS\\_GetRxFrame\(\)](#) is used to get Rx frame with zero copy, it will allocate new buffer to replace the buffer in BD taken by application application should free those buffers after they're used.

## Note

This function should be called after [ENET\\_QOS\\_CreateHandler\(\)](#) and buffer allocating callback function should be ready.

## Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | ENET_QOS peripheral base address.                                                            |
| <i>handle</i> | The ENET_QOS handler structure. This is the same handler pointer used in the EN-ET_QOS_Init. |

### 81.6.10 void ENET\_QOS\_RxBufferFreeAll ( ENET\_QOS\_Type \* *base*, enet\_qos\_handle\_t \* *handle* )

It's used for zero copy Rx. In zero copy Rx case, Rx buffers are dynamic. This function will free left buffers in all BDs.

## Parameters

|               |                                                                                              |
|---------------|----------------------------------------------------------------------------------------------|
| <i>base</i>   | ENET_QOS peripheral base address.                                                            |
| <i>handle</i> | The ENET_QOS handler structure. This is the same handler pointer used in the EN-ET_QOS_Init. |



**81.6.11** `void ENET_QOS_StartRxTx ( ENET_QOS_Type * base, uint8_t txRingNum,  
uint8_t rxRingNum )`

This function enable the tx/rx and starts the rx/tx DMA. This shall be set after ENET initialization and before starting to receive the data.

## Parameters

|                  |                                                                                                                                                                    |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | ENET peripheral base address.                                                                                                                                      |
| <i>rxRingNum</i> | The number of the used rx rings. It shall not be larger than the <a href="#">ENET_QOS_RING_NUM_MAX(2)</a> . If the ringNum is set with 1, the ring 0 will be used. |
| <i>txRingNum</i> | The number of the used tx rings. It shall not be larger than the <a href="#">ENET_QOS_RING_NUM_MAX(2)</a> . If the ringNum is set with 1, the ring 0 will be used. |

## Note

This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

**81.6.12 static void ENET\_QOS\_SetMII ( ENET\_QOS\_Type \* *base*,  
enet\_qos\_mii\_speed\_t *speed*, enet\_qos\_mii\_duplex\_t *duplex* ) [inline],  
[static]**

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

## Parameters

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

**81.6.13 void ENET\_QOS\_SetSMI ( ENET\_QOS\_Type \* *base*, uint32\_t *csrClock\_Hz*  
)**

## Parameters

|                    |                               |
|--------------------|-------------------------------|
| <i>base</i>        | ENET peripheral base address. |
| <i>csrClock_Hz</i> | CSR clock frequency in HZ     |

**81.6.14 static bool ENET\_QOS\_IsSMIBusy ( ENET\_QOS\_Type \* *base* )  
[inline], [static]**

## Parameters

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

## Returns

The status of MII Busy status.

**81.6.15** `static uint16_t ENET_QOS_ReadSMIData ( ENET_QOS_Type * base )  
[inline], [static]`

## Parameters

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

## Returns

The data read from PHY

**81.6.16** `void ENET_QOS_StartSMIRead ( ENET_QOS_Type * base, uint32_t  
phyAddr, uint32_t phyReg )`

It supports MDIO IEEE802.3 Clause 22. After send command, user needs to check whether the transmission is over with [ENET\\_QOS\\_IsSMIBusy\(\)](#).

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | ENET peripheral base address. |
| <i>phyAddr</i> | The PHY address.              |
| <i>phyReg</i>  | The PHY register.             |

**81.6.17** `void ENET_QOS_StartSMIWrite ( ENET_QOS_Type * base, uint32_t  
phyAddr, uint32_t phyReg, uint32_t data )`

It supports MDIO IEEE802.3 Clause 22. After send command, user needs to check whether the transmission is over with [ENET\\_QOS\\_IsSMIBusy\(\)](#).

## Parameters

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

### 81.6.18 void ENET\_QOS\_StartExtC45SMIWrite ( ENET\_QOS\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *device*, uint32\_t *phyReg*, uint32\_t *data* )

It supports MDIO IEEE802.3 Clause 45. After send command, user needs to check whether the transmission is over with [ENET\\_QOS\\_IsSMIBusy\(\)](#).

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | ENET peripheral base address. |
| <i>phyAddr</i> | The PHY address.              |
| <i>device</i>  | The PHY device type.          |
| <i>phyReg</i>  | The PHY register address.     |
| <i>data</i>    | The data written to PHY.      |

### 81.6.19 void ENET\_QOS\_StartExtC45SMIRead ( ENET\_QOS\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *device*, uint32\_t *phyReg* )

It supports MDIO IEEE802.3 Clause 45. After send command, user needs to check whether the transmission is over with [ENET\\_QOS\\_IsSMIBusy\(\)](#).

## Parameters

|                |                               |
|----------------|-------------------------------|
| <i>base</i>    | ENET peripheral base address. |
| <i>phyAddr</i> | The PHY address.              |
| <i>device</i>  | The PHY device type.          |
| <i>phyReg</i>  | The PHY register address.     |

### 81.6.20 static void ENET\_QOS\_SetMacAddr ( ENET\_QOS\_Type \* *base*, uint8\_t \* *macAddr*, uint8\_t *index* ) [inline], [static]

## Parameters

|                |                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                     |
| <i>macAddr</i> | The six-byte Mac address pointer. The pointer is allocated by application and input into the API. |
| <i>index</i>   | Configure macAddr to MAC_ADDRESS[index] register.                                                 |

**81.6.21 void ENET\_QOS\_GetMacAddr ( ENET\_QOS\_Type \* *base*, uint8\_t \* *macAddr*, uint8\_t *index* )**

## Parameters

|                |                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                     |
| <i>macAddr</i> | The six-byte Mac address pointer. The pointer is allocated by application and input into the API. |
| <i>index</i>   | Get macAddr from MAC_ADDRESS[index] register.                                                     |

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

## Parameters

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

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

## Parameters

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

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

#### 81.6.24 static void ENET\_QOS\_AcceptAllMulticast ( ENET\_QOS\_Type \* *base* ) [inline], [static]

Parameters

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

#### 81.6.25 static void ENET\_QOS\_RejectAllMulticast ( ENET\_QOS\_Type \* *base* ) [inline], [static]

Parameters

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

#### 81.6.26 void ENET\_QOS\_EnterPowerDown ( ENET\_QOS\_Type \* *base*, uint32\_t \* *wakeFilter* )

the remote power wake up frame and magic frame can wake up the ENET from the power down mode.

Parameters

|                   |                                                                                                                                                                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | ENET peripheral base address.                                                                                                                                                                                                               |
| <i>wakeFilter</i> | The wakeFilter provided to configure the wake up frame filter. Set the wakeFilter to NULL is not required. But if you have the filter requirement, please make sure the wakeFilter pointer shall be eight continuous 32-bits configuration. |

#### 81.6.27 static void ENET\_QOS\_ExitPowerDown ( ENET\_QOS\_Type \* *base* ) [inline], [static]

Exit from the power down mode and recover to normal work mode.

## Parameters

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

### 81.6.28 **status\_t ENET\_QOS\_EnableRxParser ( ENET\_QOS\_Type \* *base*, bool *enable* )**

## Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | ENET_QOS peripheral base address. |
| <i>enable</i> | Enable/Disable Rx parser function |

## Return values

|                                 |                              |
|---------------------------------|------------------------------|
| <i>kStatus_Success</i>          | Configure rx parser success. |
| <i>kStatus_ENET_QOS_Timeout</i> | Poll status flag timeout.    |

### 81.6.29 **void ENET\_QOS\_EnableInterrupts ( ENET\_QOS\_Type \* *base*, uint32\_t *mask* )**

This function enables the ENET interrupt according to the provided mask. The mask is a logical OR of `enet_qos_dma_interrupt_enable_t` and `enet_qos_mac_interrupt_enable_t`. For example, to enable the dma and mac interrupt, do the following.

```
* ENET_QOS_EnableInterrupts(ENET, kENET_QOS_DmaRx |
* kENET_QOS_DmaTx | kENET_QOS_MacPmt);
*
```

## Parameters

|             |                                                                                                                                                                        |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                                                                          |
| <i>mask</i> | ENET interrupts to enable. This is a logical OR of both enumeration :: <code>enet_qos_dma_interrupt_enable_t</code> and <code>enet_qos_mac_interrupt_enable_t</code> . |

### 81.6.30 **void ENET\_QOS\_DisableInterrupts ( ENET\_QOS\_Type \* *base*, uint32\_t *mask* )**

This function disables the ENET interrupt according to the provided mask. The mask is a logical OR of `enet_qos_dma_interrupt_enable_t` and `enet_qos_mac_interrupt_enable_t`. For example, to disable the

dma and mac interrupt, do the following.

```
* ENET_QOS_DisableInterrupts(ENET, kENET_QOS_DmaRx |
* kENET_QOS_DmaTx | kENET_QOS_MacPmt);
```

Parameters

|             |                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                                                                            |
| <i>mask</i> | ENET interrupts to disables. This is a logical OR of both enumeration :: <code>enet_qos_dma_interrupt_enable_t</code> and <code>enet_qos_mac_interrupt_enable_t</code> . |

**81.6.31 static uint32\_t ENET\_QOS\_GetDmaInterruptStatus ( ENET\_QOS\_Type \* *base*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                |                                                                                |
|----------------|--------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                  |
| <i>channel</i> | The DMA Channel. Shall not be larger than <code>ENET_QOS_RING_NUM_MAX</code> . |

Returns

The event status of the interrupt source. This is the logical OR of members of the enumeration :: `enet_qos_dma_interrupt_enable_t`.

**81.6.32 static void ENET\_QOS\_ClearDmaInterruptStatus ( ENET\_QOS\_Type \* *base*, uint8\_t *channel*, uint32\_t *mask* ) [inline], [static]**

Parameters

|                |                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                                                              |
| <i>channel</i> | The DMA Channel. Shall not be larger than <code>ENET_QOS_RING_NUM_MAX</code> .                                                             |
| <i>mask</i>    | The interrupt status to be cleared. This is the logical OR of members of the enumeration :: <code>enet_qos_dma_interrupt_enable_t</code> . |

**81.6.33 static uint32\_t ENET\_QOS\_GetMacInterruptStatus ( ENET\_QOS\_Type \* *base* ) [inline], [static]**



## Parameters

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

## Returns

The event status of the interrupt source. Use the enum in `enet_qos_mac_interrupt_enable_t` and right shift `ENET_QOS_MACINT_ENUM_OFFSET` to mask the returned value to get the exact interrupt status.

### 81.6.34 void ENET\_QOS\_ClearMacInterruptStatus ( ENET\_QOS\_Type \* *base*, uint32\_t *mask* )

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

```
* ENET_QOS_ClearMacInterruptStatus (ENET, kENET_QOS_MacPmt);
*
```

## Parameters

|             |                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ENET peripheral base address.                                                                                                               |
| <i>mask</i> | ENET interrupt source to be cleared. This is the logical OR of members of the enumeration :: <code>enet_qos_mac_interrupt_enable_t</code> . |

### 81.6.35 static bool ENET\_QOS\_IsTxDescriptorDmaOwn ( enet\_qos\_tx\_bd\_struct\_t \* *txDesc* ) [inline], [static]

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>txDesc</i> | The given tx descriptor. |
|---------------|--------------------------|

## Return values

|             |                                                                 |
|-------------|-----------------------------------------------------------------|
| <i>True</i> | the dma own tx descriptor, false application own tx descriptor. |
|-------------|-----------------------------------------------------------------|

**81.6.36 void ENET\_QOS\_SetupTxDescriptor ( enet\_qos\_tx\_bd\_struct\_t \* *txDesc*, void \* *buffer1*, uint32\_t *bytes1*, void \* *buffer2*, uint32\_t *bytes2*, uint32\_t *framelen*, bool *intEnable*, bool *tsEnable*, enet\_qos\_desc\_flag *flag*, uint8\_t *slotNum* )**

This function is a low level functional API to setup or prepare a given tx descriptor.

Parameters

|                  |                                                                      |
|------------------|----------------------------------------------------------------------|
| <i>txDesc</i>    | The given tx descriptor.                                             |
| <i>buffer1</i>   | The first buffer address in the descriptor.                          |
| <i>bytes1</i>    | The bytes in the first buffer.                                       |
| <i>buffer2</i>   | The second buffer address in the descriptor.                         |
| <i>bytes2</i>    | The bytes in the second buffer.                                      |
| <i>framelen</i>  | The length of the frame to be transmitted.                           |
| <i>intEnable</i> | Interrupt enable flag.                                               |
| <i>tsEnable</i>  | The timestamp enable.                                                |
| <i>flag</i>      | The flag of this tx descriptor, <a href="#">enet_qos_desc_flag</a> . |
| <i>slotNum</i>   | The slot num used for AV only.                                       |

Note

This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required. Transmit buffers are 'zero-copy' buffers, so the buffer must remain in memory until the packet has been fully transmitted. The buffers should be free or requeued in the transmit interrupt irq handler.

**81.6.37 static void ENET\_QOS\_UpdateTxDescriptorTail ( ENET\_QOS\_Type \* *base*, uint8\_t *channel*, uint32\_t *txDescTailAddrAlign* ) [inline], [static]**

This function is a low level functional API to update the tx descriptor tail. This is called after you setup a new tx descriptor to update the tail pointer to make the new descriptor accessible by DMA.

## Parameters

|                             |                                  |
|-----------------------------|----------------------------------|
| <i>base</i>                 | ENET peripheral base address.    |
| <i>channel</i>              | The tx DMA channel.              |
| <i>txDescTail-AddrAlign</i> | The new tx tail pointer address. |

**81.6.38 static void ENET\_QOS\_UpdateRxDescriptorTail ( ENET\_QOS\_Type \* *base*, uint8\_t *channel*, uint32\_t *rxDescTailAddrAlign* ) [inline], [static]**

This function is a low level functional API to update the the rx descriptor tail. This is called after you setup a new rx descriptor to update the tail pointer to make the new descriptor accessible by DMA and to anouse the rx poll command for DMA.

## Parameters

|                             |                                  |
|-----------------------------|----------------------------------|
| <i>base</i>                 | ENET peripheral base address.    |
| <i>channel</i>              | The rx DMA channel.              |
| <i>rxDescTail-AddrAlign</i> | The new rx tail pointer address. |

**81.6.39 static uint32\_t ENET\_QOS\_GetRxDescriptor ( enet\_qos\_rx\_bd\_struct\_t \* *rxDesc* ) [inline], [static]**

This function is a low level functional API to get the the status flag from a given rx descriptor.

## Parameters

|               |                          |
|---------------|--------------------------|
| <i>rxDesc</i> | The given rx descriptor. |
|---------------|--------------------------|

## Return values

|            |                                                           |
|------------|-----------------------------------------------------------|
| <i>The</i> | RDES3 regions for write-back format rx buffer descriptor. |
|------------|-----------------------------------------------------------|

## Note

This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

**81.6.40 void ENET\_QOS\_UpdateRxDescriptor ( enet\_qos\_rx\_bd\_struct\_t \* *rxDesc*,  
void \* *buffer1*, void \* *buffer2*, bool *intEnable*, bool *doubleBuffEnable* )**

This function is a low level functional API to Updates the buffers and the own status for a given rx descriptor.

## Parameters

|                          |                                              |
|--------------------------|----------------------------------------------|
| <i>rxDesc</i>            | The given rx descriptor.                     |
| <i>buffer1</i>           | The first buffer address in the descriptor.  |
| <i>buffer2</i>           | The second buffer address in the descriptor. |
| <i>intEnable</i>         | Interrupt enable flag.                       |
| <i>doubleBuff-Enable</i> | The double buffer enable flag.               |

## Note

This must be called after all the ENET initialization. And should be called when the ENET receive/transmit is required.

### 81.6.41 **status\_t ENET\_QOS\_ConfigureRxParser ( ENET\_QOS\_Type \* *base*, enet\_qos\_rxp\_config\_t \* *rxpConfig*, uint16\_t *entryCount* )**

This function is used to configure the flexible rx parser table.

## Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>base</i>       | ENET peripheral base address..       |
| <i>rxpConfig</i>  | The rx parser configuration pointer. |
| <i>entryCount</i> | The rx parser entry count.           |

## Return values

|                                  |                              |
|----------------------------------|------------------------------|
| <i>kStatus_Success</i>           | Configure rx parser success. |
| <i>kStatus_ENET_QOS_-Timeout</i> | Poll status flag timeout.    |

### 81.6.42 **status\_t ENET\_QOS\_ReadRxParser ( ENET\_QOS\_Type \* *base*, enet\_qos\_rxp\_config\_t \* *rxpConfig*, uint16\_t *entryIndex* )**

This function is used to read flexible rx parser configuration at specified index.

## Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>base</i>       | ENET peripheral base address..                   |
| <i>rxpConfig</i>  | The rx parser configuration pointer.             |
| <i>entryIndex</i> | The rx parser entry index to read, start from 0. |

## Return values

|                                 |                              |
|---------------------------------|------------------------------|
| <i>kStatus_Success</i>          | Configure rx parser success. |
| <i>kStatus_ENET_QOS_Timeout</i> | Poll status flag timeout.    |

### 81.6.43 **status\_t ENET\_QOS\_EstProgramGcl ( ENET\_QOS\_Type \* *base*, enet\_qos\_est\_gcl\_t \* *gcl*, uint32\_t *ptpClk\_Hz* )**

This function is used to program the Enhanced Scheduled Transmisson. (IEEE802.1Qbv)

## Parameters

|                  |                                             |
|------------------|---------------------------------------------|
| <i>base</i>      | ENET peripheral base address..              |
| <i>gcl</i>       | Pointer to the Gate Control List structure. |
| <i>ptpClk_Hz</i> | frequency of the PTP clock.                 |

### 81.6.44 **status\_t ENET\_QOS\_EstReadGcl ( ENET\_QOS\_Type \* *base*, enet\_qos\_est\_gcl\_t \* *gcl*, uint32\_t *listLen*, bool *hwList* )**

This function is used to read the Enhanced Scheduled Transmisson list. (IEEE802.1Qbv)

## Parameters

|                |                                                       |
|----------------|-------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address..                        |
| <i>gcl</i>     | Pointer to the Gate Control List structure.           |
| <i>listLen</i> | length of the provided opList array in gcl structure. |
| <i>hwList</i>  | Boolean if True read HW list, false read SW list.     |

**81.6.45**   **static void ENET\_QOS\_FpeEnable ( ENET\_QOS\_Type \* *base* )**  
          **[inline], [static]**

This function is used to enable frame preemption. (IEEE802.1Qbu)

## Parameters

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

**81.6.46 static void ENET\_QOS\_FpeDisable ( ENET\_QOS\_Type \* *base* )  
[inline], [static]**

This function is used to disable frame preemption. (IEEE802.1Qbu)

## Parameters

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

**81.6.47 static void ENET\_QOS\_FpeConfigPreemptable ( ENET\_QOS\_Type \* *base*,  
uint8\_t *queueMask* ) [inline], [static]**

This function is used to configure the preemptable queues. (IEEE802.1Qbu)

## Parameters

|                  |                                                                                              |
|------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>      | ENET peripheral base address..                                                               |
| <i>queueMask</i> | bitmask representing queues to set in preemptable mode. The N-th bit represents the queue N. |

**81.6.48 void ENET\_QOS\_AVBConfigure ( ENET\_QOS\_Type \* *base*, const  
enet\_qos\_cbs\_config\_t \* *config*, uint8\_t *queueIndex* )**

ENET\_QOS AVB feature configuration, set transmit bandwidth. This API is called when the AVB feature is required.

## Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>base</i>       | ENET_QOS peripheral base address.                 |
| <i>config</i>     | The ENET_QOS AVB feature configuration structure. |
| <i>queueIndex</i> | ENET_QOS queue index.                             |



**81.6.49** void ENET\_QOS\_GetStatistics ( ENET\_QOS\_Type \* *base*,  
enet\_qos\_transfer\_stats\_t \* *statistics* )

## Parameters

|                   |                                   |
|-------------------|-----------------------------------|
| <i>base</i>       | ENET_QOS peripheral base address. |
| <i>statistics</i> | The statistics structure pointer. |

**81.6.50 void ENET\_QOS\_CreateHandler ( ENET\_QOS\_Type \* *base*, enet\_qos\_handle\_t \* *handle*, enet\_qos\_config\_t \* *config*, enet\_qos\_buffer\_config\_t \* *bufferConfig*, enet\_qos\_callback\_t *callback*, void \* *userData* )**

This is a transactional API and it's provided to store all data which are needed during the whole transactional process. This API should not be used when you use functional APIs to do data tx/rx. This is function will store many data/flag for transactional use, so all configure API such as [ENET\\_QOS\\_Init\(\)](#), [ENET\\_QOS\\_DescriptorInit\(\)](#), [ENET\\_QOS\\_EnableInterrupts\(\)](#) etc.

## Note

as our transactional transmit API use the zero-copy transmit buffer. so there are two thing we emphasize here:

1. tx buffer free/requeue for application should be done in the tx interrupt handler. Please set callback: kENET\_QOS\_TxIntEvent with tx buffer free/requeue process APIs.
2. the tx interrupt is forced to open.

## Parameters

|                     |                               |
|---------------------|-------------------------------|
| <i>base</i>         | ENET peripheral base address. |
| <i>handle</i>       | ENET handler.                 |
| <i>config</i>       | ENET configuration.           |
| <i>bufferConfig</i> | ENET buffer configuration.    |
| <i>callback</i>     | The callback function.        |
| <i>userData</i>     | The application data.         |

**81.6.51 status\_t ENET\_QOS\_GetRxFrameSize ( ENET\_QOS\_Type \* *base*, enet\_qos\_handle\_t \* *handle*, uint32\_t \* *length*, uint8\_t *channel* )**

This function gets a received frame size from the ENET buffer descriptors.

## Note

The FCS of the frame is automatically removed by MAC and the size is the length without the FCS. After calling [ENET\\_QOS\\_GetRxFrameSize](#), [ENET\\_QOS\\_ReadFrame\(\)](#) should be called to update the receive buffers If the result is not "kStatus\_ENET\_QOS\_RxFrameEmpty".

## Parameters

|                |                                                                                         |
|----------------|-----------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                           |
| <i>handle</i>  | The ENET handler structure. This is the same handler pointer used in the ENET_QOS_Init. |
| <i>length</i>  | The length of the valid frame received.                                                 |
| <i>channel</i> | The DMAC channel for the rx.                                                            |

## Return values

|                                       |                                                                                                                                                          |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_ENET_QOS_Rx-FrameEmpty</i> | No frame received. Should not call ENET_QOS_ReadFrame to read frame.                                                                                     |
| <i>kStatus_ENET_QOS_Rx-FrameError</i> | Data error happens. <a href="#">ENET_QOS_ReadFrame</a> should be called with NULL data and NULL length to update the receive buffers.                    |
| <i>kStatus_Success</i>                | Receive a frame Successfully then the <a href="#">ENET_QOS_ReadFrame</a> should be called with the right data buffer and the captured data length input. |

**81.6.52** `status_t ENET_QOS_ReadFrame ( ENET_QOS_Type * base,  
enet_qos_handle_t * handle, uint8_t * data, uint32_t length, uint8_t  
channel, enet_qos_ptp_time_t * ts )`

This function reads a frame from the ENET DMA descriptors. The ENET\_QOS\_GetRxFrameSize should be used to get the size of the prepared data buffer. For example use rx dma channel 0:

```
* uint32_t length;
* enet_qos_handle_t g_handle;
* status = ENET_QOS_GetRxFrameSize(&g_handle, &length, 0);
* if (length != 0)
* {
* uint8_t *data = memory allocate interface;
* if (!data)
* {
* ENET_QOS_ReadFrame(ENET, &g_handle, NULL, 0, 0);
* }
* else
* {
* status = ENET_QOS_ReadFrame(ENET, &g_handle, data, length, 0);
* }
* }
* else if (status == kStatus_ENET_QOS_RxFrameError)
* {
* ENET_QOS_ReadFrame(ENET, &g_handle, NULL, 0, 0);
* }
*
```

## Parameters

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                                      |
| <i>handle</i>  | The ENET handler structure. This is the same handler pointer used in the ENET_QOS_Init.            |
| <i>data</i>    | The data buffer provided by user to store the frame which memory size should be at least "length". |
| <i>length</i>  | The size of the data buffer which is still the length of the received frame.                       |
| <i>channel</i> | The rx DMA channel. shall not be larger than 2.                                                    |
| <i>ts</i>      | Pointer to the structure <a href="#">enet_qos_ptp_time_t</a> to save frame timestamp.              |

## Returns

The execute status, successful or failure.

**81.6.53** `status_t ENET_QOS_SendFrame ( ENET_QOS_Type * base,  
enet_qos_handle_t * handle, uint8_t * data, uint32_t length, uint8_t  
channel, bool isNeedTs, void * context )`

## Note

The CRC is automatically appended to the data. Input the data to send without the CRC.

## Parameters

|                 |                                                                                       |
|-----------------|---------------------------------------------------------------------------------------|
| <i>base</i>     | ENET peripheral base address.                                                         |
| <i>handle</i>   | The ENET handler pointer. This is the same handler pointer used in the ENET_QOS_Init. |
| <i>data</i>     | The data buffer provided by user to be send.                                          |
| <i>length</i>   | The length of the data to be send.                                                    |
| <i>channel</i>  | Channel to send the frame, same with queue index.                                     |
| <i>isNeedTs</i> | True to enable timestamp save for the frame                                           |
| <i>context</i>  | pointer to user context to be kept in the tx dirty frame information.                 |

Return values

|                                     |                                                                                                                                                                                                                                                    |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>              | Send frame succeed.                                                                                                                                                                                                                                |
| <i>kStatus_ENET_QOS_TxFrameBusy</i> | Transmit buffer descriptor is busy under transmission. The transmit busy happens when the data send rate is over the MAC capacity. The waiting mechanism is recommended to be added after each call return with kStatus-<br>_ENET_QOS_TxFrameBusy. |

#### 81.6.54 void ENET\_QOS\_ReclaimTxDescriptor ( ENET\_QOS\_Type \* *base*, enet\_qos\_handle\_t \* *handle*, uint8\_t *channel* )

This function is used to update the tx descriptor status and store the tx timestamp when the 1588 feature is enabled. This is called by the transmit interrupt IRQ handler after the complete of a frame transmission.

Parameters

|                |                                                                                       |
|----------------|---------------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                         |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_QOS_Init. |
| <i>channel</i> | The tx DMA channel.                                                                   |

#### 81.6.55 void ENET\_QOS\_CommonIRQHandler ( ENET\_QOS\_Type \* *base*, enet\_qos\_handle\_t \* *handle* )

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | ENET peripheral base address. |
| <i>handle</i> | The ENET handler pointer.     |

#### 81.6.56 void ENET\_QOS\_SetISRHandler ( ENET\_QOS\_Type \* *base*, enet\_qos\_isr\_t *ISRHandler* )

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | ENET peripheral base address. |
| <i>ISRHandler</i> | The handler to install.       |

**81.6.57** `status_t ENET_QOS_Ptp1588CorrectTimerInCoarse ( ENET_QOS_Type  
* base, enet_qos_systime_op operation, uint32_t second, uint32_t  
nanosecond )`

## Parameters

|                   |                                                           |
|-------------------|-----------------------------------------------------------|
| <i>base</i>       | ENET peripheral base address.                             |
| <i>operation</i>  | The system time operation, refer to "enet_qos_systime_op" |
| <i>second</i>     | The correction second.                                    |
| <i>nanosecond</i> | The correction nanosecond.                                |

### 81.6.58 **status\_t ENET\_QOS\_Ptp1588CorrectTimerInFine ( ENET\_QOS\_Type \* *base*, uint32\_t *addend* )**

## Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | ENET peripheral base address.                 |
| <i>addend</i> | The addend value to be set in the fine method |

## Note

Should take refer to the chapter "System time correction" and see the description for the "fine correction method".

### 81.6.59 **static uint32\_t ENET\_QOS\_Ptp1588GetAddend ( ENET\_QOS\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | ENET peripheral base address. |
|-------------|-------------------------------|

## Returns

The addend value.

### 81.6.60 **void ENET\_QOS\_Ptp1588GetTimerNoIRQDisable ( ENET\_QOS\_Type \* *base*, uint64\_t \* *second*, uint32\_t \* *nanosecond* )**

## Parameters

|                   |                                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | ENET peripheral base address.                                                                                          |
| <i>second</i>     | The PTP 1588 system timer second.                                                                                      |
| <i>nanosecond</i> | The PTP 1588 system timer nanosecond. For the unit of the nanosecond is 1ns. so the nanosecond is the real nanosecond. |

**81.6.61** `static status_t ENET_Ptp1588PpsControl ( ENET_QOS_Type * base,  
enet_qos_ptp_pps_instance_t instance, enet_qos_ptp_pps_trgt_mode_t  
trgtMode, enet_qos_ptp_pps_cmd_t cmd ) [inline], [static]`

All channels operate in flexible PPS output mode.

## Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>base</i>     | ENET peripheral base address.                   |
| <i>instance</i> | The ENET QOS PTP PPS instance.                  |
| <i>trgtMode</i> | The target time register mode.                  |
| <i>cmd</i>      | The target flexible PPS output control command. |

**81.6.62** `status_t ENET_QOS_Ptp1588PpsSetTrgtTime ( ENET_QOS_Type * base,  
enet_qos_ptp_pps_instance_t instance, uint32_t seconds, uint32_t  
nanoseconds )`

## Parameters

|                    |                                   |
|--------------------|-----------------------------------|
| <i>base</i>        | ENET QOS peripheral base address. |
| <i>instance</i>    | The ENET QOS PTP PPS instance.    |
| <i>seconds</i>     | The target seconds.               |
| <i>nanoseconds</i> | The target nanoseconds.           |

**81.6.63** `static void ENET_QOS_Ptp1588PpsSetWidth ( ENET_QOS_Type * base,  
enet_qos_ptp_pps_instance_t instance, uint32_t width ) [inline],  
[static]`



## Parameters

|                 |                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>     | ENET QOS peripheral base address.                                                                                                         |
| <i>instance</i> | The ENET QOS PTP PPS instance.                                                                                                            |
| <i>width</i>    | Signal Width. It is stored in terms of number of units of sub-second increment value. The width value must be lesser than interval value. |

**81.6.64** `static void ENET_QOS_Ptp1588PpsSetInterval ( ENET_QOS_Type * base,  
enet_qos_ptp_pps_instance_t instance, uint32_t interval ) [inline],  
[static]`

## Parameters

|                 |                                                                                          |
|-----------------|------------------------------------------------------------------------------------------|
| <i>base</i>     | ENET QOS peripheral base address.                                                        |
| <i>instance</i> | The ENET QOS PTP PPS instance.                                                           |
| <i>interval</i> | Signal Interval. It is stored in terms of number of units of sub-second increment value. |

**81.6.65** `void ENET_QOS_Ptp1588GetTimer ( ENET_QOS_Type * base, uint64_t *  
second, uint32_t * nanosecond )`

## Parameters

|                   |                                                                                                                       |
|-------------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | ENET peripheral base address.                                                                                         |
| <i>second</i>     | The PTP 1588 system timer second.                                                                                     |
| <i>nanosecond</i> | The PTP 1588 system timer nanosecond. For the unit of the nanosecond is 1ns.so the nanosecond is the real nanosecond. |

**81.6.66** `void ENET_QOS_GetTxFrame ( enet_qos_handle_t * handle,  
enet_qos_frame_info_t * txFrame, uint8_t channel )`

This function is used for PTP stack to get the timestamp captured by the ENET driver.

## Parameters

|                |                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------|
| <i>handle</i>  | The ENET handler pointer. This is the same state pointer used in ENET_QOS_Init.                          |
| <i>txFrame</i> | Input parameter, pointer to <a href="#">enet_qos_frame_info_t</a> for saving read out frame information. |
| <i>channel</i> | Channel for searching the tx frame.                                                                      |

**81.6.67** `status_t ENET_QOS_GetRxFrame ( ENET_QOS_Type * base,  
enet_qos_handle_t * handle, enet_qos_rx_frame_struct_t * rxFrame,  
uint8_t channel )`

This function will use the user-defined allocate and free callback. Every time application gets one frame through this function, driver will allocate new buffers for the BDs whose buffers have been taken by application.

## Note

This function will drop current frame and update related BDs as available for DMA if new buffers allocating fails. Application must provide a memory pool including at least BD number + 1 buffers(+2 if enable double buffer) to make this function work normally. If user calls this function in Rx interrupt handler, be careful that this function makes Rx BD ready with allocating new buffer(normal) or updating current BD(out of memory). If there's always new Rx frame input, Rx interrupt will be triggered forever. Application need to disable Rx interrupt according to specific design in this case.

## Parameters

|                |                                                                                   |
|----------------|-----------------------------------------------------------------------------------|
| <i>base</i>    | ENET peripheral base address.                                                     |
| <i>handle</i>  | The ENET handler pointer. This is the same handler pointer used in the ENET_Init. |
| <i>rxFrame</i> | The received frame information structure provided by user.                        |
| <i>channel</i> | Channel for searching the rx frame.                                               |

## Return values

|                                       |                                                                 |
|---------------------------------------|-----------------------------------------------------------------|
| <i>kStatus_Success</i>                | Succeed to get one frame and allocate new memory for Rx buffer. |
| <i>kStatus_ENET_QOS_Rx-FrameEmpty</i> | There's no Rx frame in the BD.                                  |
| <i>kStatus_ENET_QOS_Rx-FrameError</i> | There's issue in this receiving.                                |
| <i>kStatus_ENET_QOS_Rx-FrameDrop</i>  | There's no new buffer memory for BD, drop this frame.           |

## **81.7 Variable Documentation**

### **81.7.1 `const clock_ip_name_t s_enetqosClock[]`**

## Chapter 82

### lee

#### 82.1 Overview

##### Data Structures

- struct [iee\\_config\\_t](#)  
*IEE configuration structure. [More...](#)*

##### Enumerations

- enum [iee\\_region\\_t](#) {  
    [kIEE\\_Region0](#) = 0U,  
    [kIEE\\_Region1](#) = 1U,  
    [kIEE\\_Region2](#) = 2U,  
    [kIEE\\_Region3](#) = 3U,  
    [kIEE\\_Region4](#) = 4U,  
    [kIEE\\_Region5](#) = 5U,  
    [kIEE\\_Region6](#) = 6U,  
    [kIEE\\_Region7](#) = 7U }  
    *IEE region.*
- enum [iee\\_aes\\_bypass\\_t](#) {  
    [kIEE\\_AesUseMdField](#) = 0U,  
    [kIEE\\_AesBypass](#) = 1U }  
    *IEE AES enablement/bypass.*
- enum [iee\\_aes\\_mode\\_t](#) {  
    [kIEE\\_ModeNone](#) = 0U,  
    [kIEE\\_ModeAesXTS](#) = 1U,  
    [kIEE\\_ModeAesCTRWAddress](#) = 2U,  
    [kIEE\\_ModeAesCTRWOAddress](#) = 3U,  
    [kIEE\\_ModeAesCTRkeystream](#) = 4U }  
    *IEE AES mode.*
- enum [iee\\_aes\\_key\\_size\\_t](#) {  
    [kIEE\\_AesCTR128XTS256](#) = 0U,  
    [kIEE\\_AesCTR256XTS512](#) = 1U }  
    *IEE AES key size.*
- enum [iee\\_aes\\_key\\_num\\_t](#) {  
    [kIEE\\_AesKey1](#) = 1U,  
    [kIEE\\_AesKey2](#) = 2U }  
    *IEE AES ke number.*

##### Functions

- void [IEE\\_Init](#) (IEE\_Type \*base)

- *Resets IEE module to factory default values.*
- void [IEE\\_GetDefaultConfig](#) ([iee\\_config\\_t](#) \*config)  
*Loads default values to the IEE configuration structure.*
- void [IEE\\_SetRegionConfig](#) ([IEE\\_Type](#) \*base, [iee\\_region\\_t](#) region, [iee\\_config\\_t](#) \*config)  
*Sets the IEE module according to the configuration structure.*
- [status\\_t](#) [IEE\\_SetRegionKey](#) ([IEE\\_Type](#) \*base, [iee\\_region\\_t](#) region, [iee\\_aes\\_key\\_num\\_t](#) keyNum, [const](#) [uint8\\_t](#) \*key, [size\\_t](#) keySize)  
*Sets the IEE module key.*
- [static](#) [uint32\\_t](#) [IEE\\_GetOffset](#) ([uint32\\_t](#) addressIee, [uint32\\_t](#) addressMemory)  
*Computes IEE offset to be set for specifed memory location.*
- void [IEE\\_LockRegionConfig](#) ([IEE\\_Type](#) \*base, [iee\\_region\\_t](#) region)  
*Lock the IEE region configuration.*

## Driver version

- [#define](#) [FSL\\_IEE\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))  
*IEE driver version.*

## 82.2 Data Structure Documentation

### 82.2.1 struct iee\_config\_t

#### Data Fields

- [iee\\_aes\\_bypass\\_t](#) bypass  
*AES encryption/decryption bypass.*
- [iee\\_aes\\_mode\\_t](#) mode  
*AES mode.*
- [iee\\_aes\\_key\\_size\\_t](#) keySize  
*size of AES key*
- [uint32\\_t](#) [pageOffset](#)  
*Offset to physical memory location from IEE start address.*

## 82.3 Macro Definition Documentation

### 82.3.1 [#define](#) [FSL\\_IEE\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))

Version 2.1.1.

Current version: 2.1.1

Change log:

- Version 2.0.0
  - Initial version
- Version 2.1.0
  - Add region lock function [IEE\\_LockRegionConfig\(\)](#) and driver clock control
- Version 2.1.1
  - Fixed MISRA issues.

## 82.4 Enumeration Type Documentation

### 82.4.1 enum iee\_region\_t

Enumerator

***kIEE\_Region0*** IEE region 0.  
***kIEE\_Region1*** IEE region 1.  
***kIEE\_Region2*** IEE region 2.  
***kIEE\_Region3*** IEE region 3.  
***kIEE\_Region4*** IEE region 4.  
***kIEE\_Region5*** IEE region 5.  
***kIEE\_Region6*** IEE region 6.  
***kIEE\_Region7*** IEE region 7.

### 82.4.2 enum iee\_aes\_bypass\_t

Enumerator

***kIEE\_AesUseMdField*** AES encryption/decryption enabled.  
***kIEE\_AesBypass*** AES encryption/decryption bypass.

### 82.4.3 enum iee\_aes\_mode\_t

Enumerator

***kIEE\_ModeNone*** AES NONE mode.  
***kIEE\_ModeAesXTS*** AES XTS mode.  
***kIEE\_ModeAesCTRWAddress*** CTR w address binding mode.  
***kIEE\_ModeAesCTRWOAddress*** AES CTR w/o address binding mode.  
***kIEE\_ModeAesCTRkeystream*** AES CTR keystream only.

### 82.4.4 enum iee\_aes\_key\_size\_t

Enumerator

***kIEE\_AesCTR128XTS256*** AES 128 bits (CTR), 256 bits (XTS)  
***kIEE\_AesCTR256XTS512*** AES 256 bits (CTR), 512 bits (XTS)

### 82.4.5 enum iee\_aes\_key\_num\_t

Enumerator

*kIEE\_AesKey1* AES Key 1.  
*kIEE\_AesKey2* AES Key 2.

## 82.5 Function Documentation

### 82.5.1 void IEE\_Init ( IEE\_Type \* *base* )

This function performs hardware reset of IEE module. Attributes and keys of all regions are cleared.

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | IEER peripheral address. |
|-------------|--------------------------|

### 82.5.2 void IEE\_GetDefaultConfig ( iee\_config\_t \* *config* )

Loads default values to the IEE region configuration structure. The default values are as follows.

```
* config->bypass = kIEE_AesUseMdField;
* config->mode = kIEE_ModeNone;
* config->keySize = kIEE_AesCTR128XTS256;
* config->pageOffset = 0U;
*
```

Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>config</i> | Configuration for the selected IEE region. |
|---------------|--------------------------------------------|

### 82.5.3 void IEE\_SetRegionConfig ( IEE\_Type \* *base*, iee\_region\_t *region*, iee\_config\_t \* *config* )

This function configures IEE region according to configuration structure.

Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | IEE peripheral address. |
|-------------|-------------------------|

|               |                                               |
|---------------|-----------------------------------------------|
| <i>region</i> | Selection of the IEE region to be configured. |
| <i>config</i> | Configuration for the selected IEE region.    |

#### 82.5.4 **status\_t IEE\_SetRegionKey ( IEE\_Type \* *base*, iee\_region\_t *region*, iee\_aes\_key\_num\_t *keyNum*, const uint8\_t \* *key*, size\_t *keySize* )**

This function sets specified AES key for the given region.

Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>base</i>    | IEE peripheral address.                       |
| <i>region</i>  | Selection of the IEE region to be configured. |
| <i>keyNum</i>  | Selection of AES KEY1 or KEY2.                |
| <i>key</i>     | AES key.                                      |
| <i>keySize</i> | Size of AES key.                              |

#### 82.5.5 **static uint32\_t IEE\_GetOffset ( uint32\_t *addressIee*, uint32\_t *addressMemory* ) [inline], [static]**

This function calculates offset that must be set for IEE region to access physical memory location.

Parameters

|                       |                                      |
|-----------------------|--------------------------------------|
| <i>addressIee</i>     | Address of IEE peripheral.           |
| <i>address-Memory</i> | Address of physical memory location. |

#### 82.5.6 **void IEE\_LockRegionConfig ( IEE\_Type \* *base*, iee\_region\_t *region* )**

This function locks IEE region registers for Key, Offset and Attribute. Only system reset can clear the Lock bit.

Parameters

---



|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | IEE peripheral address.                   |
| <i>region</i> | Selection of the IEE region to be locked. |

## Chapter 83

### leer

#### 83.1 Overview

##### Enumerations

- enum `iee_apc_region_t` {  
    `kIEE_APC_Region0` = 0U,  
    `kIEE_APC_Region1` = 1U,  
    `kIEE_APC_Region2` = 2U,  
    `kIEE_APC_Region3` = 3U,  
    `kIEE_APC_Region4` = 4U,  
    `kIEE_APC_Region5` = 5U,  
    `kIEE_APC_Region6` = 6U,  
    `kIEE_APC_Region7` = 7U }  
    *APC IEE regions.*
- enum `iee_apc_domain_t` {  
    `kIEE_APC_Domain0` = 0U,  
    `kIEE_APC_Domain1` = 1U }  
    *APC IEE domains.*

##### Functions

- void `IEE_APC_GlobalEnable` (IEE\_APC\_Type \*base)  
    *Enable the APC IEE Region setting.*
- void `IEE_APC_GlobalDisable` (IEE\_APC\_Type \*base)  
    *Disables the APC IEE Region setting.*
- `status_t` `IEE_APC_SetRegionConfig` (IEE\_APC\_Type \*base, `iee_apc_region_t` region, uint32\_t startAddr, uint32\_t endAddr)  
    *Sets the APC IEE Memory Region Descriptors.*
- `status_t` `IEE_APC_LockRegionConfig` (IEE\_APC\_Type \*base, `iee_apc_region_t` region, `iee_apc_domain_t` domain)  
    *Lock the LPSR GPR and APC IEE configuration.*
- void `IEE_APC_RegionEnable` (IEE\_APC\_Type \*base, `iee_apc_region_t` region)  
    *Enable the IEE encryption/decryption and can lock this setting.*

##### Driver version

- #define `FSL_IEE_APC_DRIVER_VERSION` (MAKE\_VERSION(2, 0, 1))  
    *IEE\_APC driver version.*

#### 83.2 Macro Definition Documentation

### 83.2.1 #define FSL\_IEE\_APC\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Version 2.0.1.

Current version: 2.0.1

Change log:

- Version 2.0.0
  - Initial version
- Version 2.0.1
  - Fixed MISRA issues.

## 83.3 Enumeration Type Documentation

### 83.3.1 enum iee\_apc\_region\_t

Enumerator

***kIEE\_APC\_Region0*** APC IEE region 0.  
***kIEE\_APC\_Region1*** APC IEE region 1.  
***kIEE\_APC\_Region2*** APC IEE region 2.  
***kIEE\_APC\_Region3*** APC IEE region 3.  
***kIEE\_APC\_Region4*** APC IEE region 4.  
***kIEE\_APC\_Region5*** APC IEE region 5.  
***kIEE\_APC\_Region6*** APC IEE region 6.  
***kIEE\_APC\_Region7*** APC IEE region 7.

### 83.3.2 enum iee\_apc\_domain\_t

Enumerator

***kIEE\_APC\_Domain0*** APC IEE region 0.  
***kIEE\_APC\_Domain1*** APC IEE region 1.

## 83.4 Function Documentation

### 83.4.1 void IEE\_APC\_GlobalEnable ( IEE\_APC\_Type \* *base* )

This function enables IOMUXC LPSR GPR and APC IEE for setting the region.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | APC IEE peripheral address. |
|-------------|-----------------------------|

### 83.4.2 void IEE\_APC\_GlobalDisable ( IEE\_APC\_Type \* *base* )

This function disables IOMUXC LPSR GPR and APC IEE for setting the region.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | APC IEE peripheral address. |
|-------------|-----------------------------|

### 83.4.3 status\_t IEE\_APC\_SetRegionConfig ( IEE\_APC\_Type \* *base*, iee\_apc\_region\_t *region*, uint32\_t *startAddr*, uint32\_t *endAddr* )

This function configures APC IEE Memory Region Descriptor according to region configuration structure.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | APC IEE peripheral address.                               |
| <i>region</i>    | Selection of the APC IEE region to be configured.         |
| <i>startAddr</i> | Start encryption address for the selected APC IEE region. |
| <i>endAddr</i>   | End encryption address for the selected APC IEE region.   |

### 83.4.4 status\_t IEE\_APC\_LockRegionConfig ( IEE\_APC\_Type \* *base*, iee\_apc\_region\_t *region*, iee\_apc\_domain\_t *domain* )

This function locks writting to IOMUXC LPSR GPR and APC IEE encryption region setting registers. Only system reset can clear the LPSR GPR and APC IEE-RDC\_D0/1 Lock bit

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | APC IEE peripheral address.                   |
| <i>region</i> | Selection of the APC IEE region to be locked. |

|               |  |
|---------------|--|
| <i>domain</i> |  |
|---------------|--|

#### 83.4.5 void IEE\_APC\_RegionEnable ( IEE\_APC\_Type \* *base*, iee\_apc\_region\_t *region* )

This function enables encryption/decryption by writting to IOMUXC LPSR GPR.

Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | APC IEE peripheral address.                    |
| <i>region</i> | Selection of the APC IEE region to be enabled. |

# Chapter 84

## Key\_manager

### 84.1 Overview

#### Data Structures

- struct [domain\\_slot\\_config\\_t](#)  
*Key Manager slot configuration structure. [More...](#)*

#### Functions

- [status\\_t KEYMGR\\_MasterKeyControll](#) (KEY\_MANAGER\_Type \*base, keymgr\_select\_t select, keymgr\_lock\_t lock)  
*Configures Master key settings.*
- [status\\_t KEYMGR\\_OTFAD1KeyControll](#) (KEY\_MANAGER\_Type \*base, keymgr\_select\_t select, keymgr\_lock\_t lock)  
*Configures OTFAD1 key settings.*
- [status\\_t KEYMGR\\_OTFAD2KeyControll](#) (KEY\_MANAGER\_Type \*base, keymgr\_select\_t select, keymgr\_lock\_t lock)  
*Configures OTFAD2 key settings.*
- void [KEYMGR\\_IEEKeyReload](#) (KEY\_MANAGER\_Type \*base)  
*Restart load key signal for IEE.*
- void [KEYMGR\\_PUFKeyLock](#) (KEY\_MANAGER\_Type \*base, keymgr\_lock\_t lock)  
*Lock the key select from PUF.*
- [status\\_t KEYMGR\\_SlotControl](#) (KEY\_MANAGER\_Type \*base, [domain\\_slot\\_config\\_t](#) \*config, keymgr\_slot\_t slot)  
*Configures Slot Domain control.*
- void [KEYMGR\\_Init](#) (KEY\_MANAGER\_Type \*base)  
*Resets Key Manager module to factory default values.*
- [status\\_t KEYMGR\\_GetDefaultConfig](#) ([domain\\_slot\\_config\\_t](#) \*config)  
*Sets the default configuration of Key manager slot.*

#### Driver version

- #define [FSL\\_KEYMGR\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 2))  
*Key Manager driver version.*

### 84.2 Data Structure Documentation

#### 84.2.1 struct domain\_slot\_config\_t

##### Data Fields

- keymgr\_lock\_t [lockControl](#)

- *Lock control register of slot.*  
keymgr\_allow\_t allowUser  
*Allow user write access to domain control register or domain register.*
- keymgr\_allow\_t allowNonSecure  
*Allow non-secure write access to domain control register or domain register.*
- keymgr\_lock\_t lockList  
*Lock whitelist.*
- uint8\_t whiteList  
*Domains that on the Whitelist can change given slot.*

## Field Documentation

(1) keymgr\_lock\_t domain\_slot\_config\_t::lockControl

(2) keymgr\_allow\_t domain\_slot\_config\_t::allowUser

(3) keymgr\_allow\_t domain\_slot\_config\_t::allowNonSecure

(4) keymgr\_lock\_t domain\_slot\_config\_t::lockList

SLOTx\_CTRL[WHITE\_LIST] cannot be changed.

(5) uint8\_t domain\_slot\_config\_t::whiteList

Each field represents one domain. Bit0~Bit3 represent DOMAIN0~DOMAIN3 respectively.

## 84.3 Macro Definition Documentation

### 84.3.1 #define FSL\_KEYMGR\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 2))

Version 2.0.2.

Current version: 2.0.2

Change log:

- Version 2.0.2
  - Fix MISRA-2012 issues
- Version 2.0.1
  - Fix MISRA-2012 issues
- Version 2.0.0
  - Initial version

## 84.4 Function Documentation

### 84.4.1 status\_t KEYMGR\_MasterKeyControl ( KEY\_MANAGER\_Type \* base, keymgr\_select\_t select, keymgr\_lock\_t lock )

This function configures Key Manager's setting for Master key.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Key Manager peripheral address. |
| <i>select</i> | select source for Master key.   |
| <i>lock</i>   | setting for lock Master key.    |

## Returns

status of Master key control operation

#### 84.4.2 **status\_t KEYMGR\_OTFAD1KeyControl ( KEY\_MANAGER\_Type \* *base*, keymgr\_select\_t *select*, keymgr\_lock\_t *lock* )**

This function configures Key Manager's setting for OTFAD1 key.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Key Manager peripheral address. |
| <i>select</i> | select source for OTFAD1 key.   |
| <i>lock</i>   | setting for lock OTFAD1 key.    |

## Returns

status of OTFAD1 key control operation

#### 84.4.3 **status\_t KEYMGR\_OTFAD2KeyControl ( KEY\_MANAGER\_Type \* *base*, keymgr\_select\_t *select*, keymgr\_lock\_t *lock* )**

This function configures Key Manager's setting for OTFAD2 key.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Key Manager peripheral address. |
| <i>select</i> | select source for OTFAD2 key.   |



|             |                              |
|-------------|------------------------------|
| <i>lock</i> | setting for lock OTFAD2 key. |
|-------------|------------------------------|

Returns

status of OTFAD2 key control operation

#### 84.4.4 void KEYMGR\_IEEKeyReload ( KEY\_MANAGER\_Type \* *base* )

This function generates Key Manager's restart signal for IEE key.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Key Manager peripheral address. |
|-------------|---------------------------------|

#### 84.4.5 void KEYMGR\_PUFKeyLock ( KEY\_MANAGER\_Type \* *base*, keymgr\_lock\_t *lock* )

This function locks selection of key for PUF.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Key Manager peripheral address.       |
| <i>lock</i> | Setting for selection of key for PUF. |

#### 84.4.6 status\_t KEYMGR\_SlotControl ( KEY\_MANAGER\_Type \* *base*, domain\_slot\_config\_t \* *config*, keymgr\_slot\_t *slot* )

This function configures domain slot control which locks and allows writes.

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | Key Manager peripheral address.          |
| <i>config</i> | Pointer to slot configuration structure. |
| <i>slot</i>   | Select slot to be configured.            |

Returns

status of slot control operation

#### **84.4.7 void KEYMGR\_Init ( KEY\_MANAGER\_Type \* *base* )**

This function performs hardware reset of Key Manager module.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Key Manager peripheral address. |
|-------------|---------------------------------|

#### 84.4.8 **status\_t** KEYMGR\_GetDefaultConfig ( **domain\_slot\_config\_t** \* *config* )

This function initialize Key Manager slot config structure to default values.

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>config</i> | Pointer to slot configuration structure. |
|---------------|------------------------------------------|

**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.

