MCUXpresso SDK API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKKW35AAPIRM
Rev. 0
Jun 2017

Contents

Chapter **Introduction**

Chapter **Driver errors status**

Chapter **Architectural Overview**

Chapter **Trademarks**

Chapter **ADC16: 16-bit SAR Analog-to-Digital Converter Driver**

5.1 **Overview** **11**

5.2 **Typical use case** **11**
5.2.1 Polling Configuration 11
5.2.2 Interrupt Configuration 11

5.3 **Data Structure Documentation** **15**
5.3.1 struct adc16_config_t 15
5.3.2 struct adc16_hardware_compare_config_t 16
5.3.3 struct adc16_channel_config_t 16

5.4 **Macro Definition Documentation** **17**
5.4.1 FSL_ADC16_DRIVER_VERSION 17

5.5 **Enumeration Type Documentation** **17**
5.5.1 _adc16_channel_status_flags 17
5.5.2 _adc16_status_flags 17
5.5.3 adc16_channel_mux_mode_t 17
5.5.4 adc16_clock_divider_t 18
5.5.5 adc16_resolution_t 18
5.5.6 adc16_clock_source_t 18
5.5.7 adc16_long_sample_mode_t 18
5.5.8 adc16_reference_voltage_source_t 19
5.5.9 adc16_hardware_average_mode_t 19
5.5.10 adc16_hardware_compare_mode_t 19

5.6 **Function Documentation** **19**
5.6.1 ADC16_Init 19

Contents

Section Number	Title	Page Number
5.6.2	ADC16_Deinit	20
5.6.3	ADC16_GetDefaultConfig	20
5.6.4	ADC16_DoAutoCalibration	20
5.6.5	ADC16_SetOffsetValue	21
5.6.6	ADC16_EnableDMA	21
5.6.7	ADC16_EnableHardwareTrigger	21
5.6.8	ADC16_SetChannelMuxMode	22
5.6.9	ADC16_SetHardwareCompareConfig	22
5.6.10	ADC16_SetHardwareAverage	22
5.6.11	ADC16_GetStatusFlags	23
5.6.12	ADC16_ClearStatusFlags	23
5.6.13	ADC16_SetChannelConfig	23
5.6.14	ADC16_GetChannelConversionValue	25
5.6.15	ADC16_GetChannelStatusFlags	25

Chapter CMP: Analog Comparator Driver

6.1	**Overview**	**27**
6.2	**Typical use case**	**27**
6.2.1	Polling Configuration	27
6.2.2	Interrupt Configuration	28
6.3	**Data Structure Documentation**	**30**
6.3.1	struct cmp_config_t	30
6.3.2	struct cmp_filter_config_t	31
6.3.3	struct cmp_dac_config_t	31
6.4	**Macro Definition Documentation**	**32**
6.4.1	FSL_CMP_DRIVER_VERSION	32
6.5	**Enumeration Type Documentation**	**32**
6.5.1	_cmp_interrupt_enable	32
6.5.2	_cmp_status_flags	32
6.5.3	cmp_hysteresis_mode_t	32
6.5.4	cmp_reference_voltage_source_t	32
6.6	**Function Documentation**	**33**
6.6.1	CMP_Init	33
6.6.2	CMP_Deinit	33
6.6.3	CMP_Enable	33
6.6.4	CMP_GetDefaultConfig	34
6.6.5	CMP_SetInputChannels	34
6.6.6	CMP_EnableDMA	34
6.6.7	CMP_SetFilterConfig	35

Contents

Section Number	Title	Page Number
6.6.8	CMP_SetDACConfig	35
6.6.9	CMP_EnableInterrupts	35
6.6.10	CMP_DisableInterrupts	35
6.6.11	CMP_GetStatusFlags	36
6.6.12	CMP_ClearStatusFlags	36
Chapter	**CMT: Carrier Modulator Transmitter Driver**	
7.1	**Overview**	**37**
7.2	**Clock formulas**	**37**
7.3	**Typical use case**	**37**
7.4	**Data Structure Documentation**	**40**
7.4.1	struct cmt_modulate_config_t	40
7.4.2	struct cmt_config_t	41
7.5	**Macro Definition Documentation**	**41**
7.5.1	FSL_CMT_DRIVER_VERSION	41
7.6	**Enumeration Type Documentation**	**41**
7.6.1	cmt_mode_t	41
7.6.2	cmt_primary_clkdiv_t	42
7.6.3	cmt_second_clkdiv_t	42
7.6.4	cmt_infrared_output_polarity_t	43
7.6.5	cmt_infrared_output_state_t	43
7.6.6	_cmt_interrupt_enable	43
7.7	**Function Documentation**	**43**
7.7.1	CMT_GetDefaultConfig	43
7.7.2	CMT_Init	43
7.7.3	CMT_Deinit	44
7.7.4	CMT_SetMode	44
7.7.5	CMT_GetMode	44
7.7.6	CMT_GetCMTFrequency	45
7.7.7	CMT_SetCarrirGenerateCountOne	46
7.7.8	CMT_SetCarrirGenerateCountTwo	46
7.7.9	CMT_SetModulateMarkSpace	47
7.7.10	CMT_EnableExtendedSpace	47
7.7.11	CMT_SetIroState	48
7.7.12	CMT_EnableInterrupts	49
7.7.13	CMT_DisableInterrupts	49
7.7.14	CMT_GetStatusFlags	49

Contents

Section Number	Title	Page Number
Chapter	**COP: Watchdog Driver**	
8.1	**Overview**	**51**
8.2	**Typical use case**	**51**
8.3	**Data Structure Documentation**	**52**
8.3.1	struct cop_config_t	52
8.4	**Macro Definition Documentation**	**52**
8.4.1	FSL_COP_DRIVER_VERSION	52
8.5	**Enumeration Type Documentation**	**52**
8.5.1	cop_clock_source_t	52
8.5.2	cop_timeout_cycles_t	53
8.5.3	cop_timeout_mode_t	53
8.6	**Function Documentation**	**53**
8.6.1	COP_GetDefaultConfig	53
8.6.2	COP_Init	53
8.6.3	COP_Disable	54
8.6.4	COP_Refresh	54
Chapter	**DCDC: DCDC Converter**	
9.1	**Overview**	**55**
9.2	**Function groups**	**55**
9.2.1	Initialization and deinitialization	55
9.2.2	Status	55
9.2.3	Interrupts	55
9.2.4	Misc control	55
9.3	**Application guideline**	**56**
9.3.1	Continuous mode	56
9.3.2	Target voltage adjustment	56
9.3.3	Pulsed mode	56
9.4	**Data Structure Documentation**	**59**
9.4.1	struct dcdc_low_power_config_t	59
9.4.2	struct dcdc_loop_control_config_t	60
9.4.3	struct dcdc_min_power_config_t	61
9.4.4	struct dcdc_pulsed_integrator_config_t	61
9.5	**Macro Definition Documentation**	**62**
9.5.1	FSL_DCDC_DRIVER_VERSION	62

Contents

Section Number	Title	Page Number
9.6	**Enumeration Type Documentation**	**62**
9.6.1	_dcdc_status_flags_t	62
9.6.2	_dcdc_interrupt_enable_t	62
9.6.3	_dcdc_pswitch_detect_event_t	62
9.6.4	dcdc_work_mode_t	63
9.6.5	dcdc_hysteretic_threshold_offset_value_t	63
9.6.6	dcdc_vbat_divider_t	63
9.6.7	dcdc_clock_source_t	63
9.7	**Function Documentation**	**63**
9.7.1	DCDC_Init	63
9.7.2	DCDC_Deinit	64
9.7.3	DCDC_GetStatusFlags	64
9.7.4	DCDC_ClearStatusFlags	64
9.7.5	DCDC_EnableInterrupts	64
9.7.6	DCDC_DisableInterrupts	64
9.7.7	DCDC_SetPSwitchInterruptConfig	65
9.7.8	DCDC_GetDefaultLowPowerConfig	65
9.7.9	DCDC_SetLowPowerConfig	65
9.7.10	DCDC_GetDefaultLoopControlConfig	66
9.7.11	DCDC_SetLoopControlConfig	66
9.7.12	DCDC_EnableXtalOKDetectionCircuit	66
9.7.13	DCDC_EnableOutputRangeComparator	67
9.7.14	DCDC_EnableReduceCurrent	67
9.7.15	DCDC_SetClockSource	67
9.7.16	DCDC_SetBatteryVoltageDivider	68
9.7.17	DCDC_SetBatteryMonitorValue	68
9.7.18	DCDC_DoSoftShutdown	68
9.7.19	DCDC_SetUpperLimitDutyCycleBoost	69
9.7.20	DCDC_SetUpperLimitDutyCycleBuck	69
9.7.21	DCDC_AdjustDutyCycleSwitchingTargetOutput	69
9.7.22	DCDC_LockTargetVoltage	69
9.7.23	DCDC_AdjustTargetVoltage	70
9.7.24	DCDC_GetDefaultMinPowerDefault	70
9.7.25	DCDC_SetMinPowerConfig	70
9.7.26	DCDC_GetDefaultPulsedIntegratorConfig	71
9.7.27	DCDC_SetPulsedIntegratorConfig	71
Chapter	**DMAMUX: Direct Memory Access Multiplexer Driver**	
10.1	**Overview**	**73**
10.2	**Typical use case**	**73**
10.2.1	DMAMUX Operation	73

Contents

Section Number	Title	Page Number
10.3	**Macro Definition Documentation**	**73**
10.3.1	FSL_DMAMUX_DRIVER_VERSION	73
10.4	**Function Documentation**	**74**
10.4.1	DMAMUX_Init	74
10.4.2	DMAMUX_Deinit	75
10.4.3	DMAMUX_EnableChannel	75
10.4.4	DMAMUX_DisableChannel	75
10.4.5	DMAMUX_SetSource	76
10.4.6	DMAMUX_EnablePeriodTrigger	76
10.4.7	DMAMUX_DisablePeriodTrigger	76
Chapter	**DSPI: Serial Peripheral Interface Driver**	
11.1	**Overview**	**77**
11.2	**DSPI Driver**	**78**
11.2.1	Overview	78
11.2.2	Typical use case	78
11.2.3	Data Structure Documentation	85
11.2.4	Macro Definition Documentation	92
11.2.5	Typedef Documentation	93
11.2.6	Enumeration Type Documentation	94
11.2.7	Function Documentation	98
11.3	**DSPI DMA Driver**	**116**
11.3.1	Overview	116
11.3.2	Data Structure Documentation	117
11.3.3	Typedef Documentation	120
11.3.4	Function Documentation	121
11.4	**DSPI eDMA Driver**	**126**
11.4.1	Overview	126
11.4.2	Data Structure Documentation	127
11.4.3	Typedef Documentation	130
11.4.4	Function Documentation	131
11.5	**DSPI FreeRTOS Driver**	**136**
11.5.1	Overview	136
11.5.2	Function Documentation	136
Chapter	**eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver**	
12.1	**Overview**	**139**
12.2	**Typical use case**	**139**

Contents

Section Number	Title	Page Number
12.2.1	eDMA Operation	139
12.3	**Data Structure Documentation**	**145**
12.3.1	struct edma_config_t	145
12.3.2	struct edma_transfer_config_t	145
12.3.3	struct edma_channel_Preemption_config_t	146
12.3.4	struct edma_minor_offset_config_t	146
12.3.5	struct edma_tcd_t	147
12.3.6	struct edma_handle_t	148
12.4	**Macro Definition Documentation**	**149**
12.4.1	FSL_EDMA_DRIVER_VERSION	149
12.5	**Typedef Documentation**	**149**
12.5.1	edma_callback	149
12.6	**Enumeration Type Documentation**	**149**
12.6.1	edma_transfer_size_t	149
12.6.2	edma_modulo_t	149
12.6.3	edma_bandwidth_t	150
12.6.4	edma_channel_link_type_t	150
12.6.5	_edma_channel_status_flags	150
12.6.6	_edma_error_status_flags	151
12.6.7	edma_interrupt_enable_t	151
12.6.8	edma_transfer_type_t	151
12.6.9	_edma_transfer_status	151
12.7	**Function Documentation**	**152**
12.7.1	EDMA_Init	152
12.7.2	EDMA_Deinit	153
12.7.3	EDMA_InstallTCD	153
12.7.4	EDMA_GetDefaultConfig	153
12.7.5	EDMA_ResetChannel	154
12.7.6	EDMA_SetTransferConfig	154
12.7.7	EDMA_SetMinorOffsetConfig	155
12.7.8	EDMA_SetChannelPreemptionConfig	155
12.7.9	EDMA_SetChannelLink	155
12.7.10	EDMA_SetBandWidth	157
12.7.11	EDMA_SetModulo	158
12.7.12	EDMA_EnableAsyncRequest	158
12.7.13	EDMA_EnableAutoStopRequest	158
12.7.14	EDMA_EnableChannelInterrupts	159
12.7.15	EDMA_DisableChannelInterrupts	160
12.7.16	EDMA_TcdReset	160
12.7.17	EDMA_TcdSetTransferConfig	160

Contents

Section Number	Title	Page Number
12.7.18	EDMA_TcdSetMinorOffsetConfig	161
12.7.19	EDMA_TcdSetChannelLink	161
12.7.20	EDMA_TcdSetBandWidth	162
12.7.21	EDMA_TcdSetModulo	162
12.7.22	EDMA_TcdEnableAutoStopRequest	163
12.7.23	EDMA_TcdEnableInterrupts	164
12.7.24	EDMA_TcdDisableInterrupts	164
12.7.25	EDMA_EnableChannelRequest	164
12.7.26	EDMA_DisableChannelRequest	164
12.7.27	EDMA_TriggerChannelStart	165
12.7.28	EDMA_GetRemainingMajorLoopCount	165
12.7.29	EDMA_GetErrorStatusFlags	166
12.7.30	EDMA_GetChannelStatusFlags	166
12.7.31	EDMA_ClearChannelStatusFlags	166
12.7.32	EDMA_CreateHandle	167
12.7.33	EDMA_InstallTCDMemory	168
12.7.34	EDMA_SetCallback	168
12.7.35	EDMA_PrepareTransfer	168
12.7.36	EDMA_SubmitTransfer	169
12.7.37	EDMA_StartTransfer	170
12.7.38	EDMA_StopTransfer	170
12.7.39	EDMA_AbortTransfer	170
12.7.40	EDMA_HandleIRQ	170

Chapter C90TFS Flash Driver

13.1	**Overview**	**173**
13.2	**Data Structure Documentation**	**182**
13.2.1	struct flash_execute_in_ram_function_config_t	182
13.2.2	struct flash_swap_state_config_t	182
13.2.3	struct flash_swap_ifr_field_config_t	182
13.2.4	union flash_swap_ifr_field_data_t	183
13.2.5	union pflash_protection_status_low_t	183
13.2.6	struct pflash_protection_status_t	184
13.2.7	struct flash_prefetch_speculation_status_t	184
13.2.8	struct flash_protection_config_t	184
13.2.9	struct flash_access_config_t	185
13.2.10	struct flash_operation_config_t	185
13.2.11	struct flash_config_t	186
13.3	**Macro Definition Documentation**	**188**
13.3.1	MAKE_VERSION	188
13.3.2	FSL_FLASH_DRIVER_VERSION	188
13.3.3	FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT	188

Contents

Section Number	Title	Page Number
13.3.4	FLASH_SSD_CONFIG_ENABLE_SECONDARY_FLASH_SUPPORT	188
13.3.5	FLASH_DRIVER_IS_FLASH_RESIDENT	188
13.3.6	FLASH_DRIVER_IS_EXPORTED	188
13.3.7	kStatusGroupGeneric	188
13.3.8	MAKE_STATUS	188
13.3.9	FOUR_CHAR_CODE	188
13.4	**Enumeration Type Documentation**	**188**
13.4.1	_flash_driver_version_constants	188
13.4.2	_flash_status	189
13.4.3	_flash_driver_api_keys	189
13.4.4	flash_margin_value_t	190
13.4.5	flash_security_state_t	190
13.4.6	flash_protection_state_t	190
13.4.7	flash_execute_only_access_state_t	190
13.4.8	flash_property_tag_t	190
13.4.9	_flash_execute_in_ram_function_constants	191
13.4.10	flash_read_resource_option_t	191
13.4.11	_flash_read_resource_range	191
13.4.12	_k3_flash_read_once_index	192
13.4.13	flash_flexram_function_option_t	192
13.4.14	flash_swap_function_option_t	192
13.4.15	flash_swap_control_option_t	192
13.4.16	flash_swap_state_t	193
13.4.17	flash_swap_block_status_t	193
13.4.18	flash_partition_flexram_load_option_t	193
13.4.19	flash_memory_index_t	193
13.4.20	flash_cache_controller_index_t	193
13.4.21	flash_cache_clear_process_t	194
13.5	**Function Documentation**	**194**
13.5.1	FLASH_Init	194
13.5.2	FLASH_SetCallback	194
13.5.3	FLASH_PrepareExecuteInRamFunctions	195
13.5.4	FLASH_EraseAll	195
13.5.5	FLASH_Erase	196
13.5.6	FLASH_EraseAllUnsecure	197
13.5.7	FLASH_EraseAllExecuteOnlySegments	198
13.5.8	FLASH_Program	198
13.5.9	FLASH_ProgramOnce	199
13.5.10	FLASH_ReadResource	200
13.5.11	FLASH_ReadOnce	201
13.5.12	FLASH_GetSecurityState	202
13.5.13	FLASH_SecurityBypass	202
13.5.14	FLASH_VerifyEraseAll	203

Contents

Section Number	Title	Page Number
13.5.15	FLASH_VerifyErase	204
13.5.16	FLASH_VerifyProgram	205
13.5.17	FLASH_VerifyEraseAllExecuteOnlySegments	206
13.5.18	FLASH_IsProtected	207
13.5.19	FLASH_IsExecuteOnly	207
13.5.20	FLASH_GetProperty	208
13.5.21	FLASH_SetProperty	208
13.5.22	FLASH_PflashSetProtection	209
13.5.23	FLASH_PflashGetProtection	210

Chapter FlexCAN: Flex Controller Area Network Driver

14.1	**Overview**	**213**
14.2	**FlexCAN Driver**	**214**
14.2.1	Overview	214
14.2.2	Typical use case	214
14.2.3	Data Structure Documentation	222
14.2.4	Macro Definition Documentation	226
14.2.5	Typedef Documentation	231
14.2.6	Enumeration Type Documentation	231
14.2.7	Function Documentation	234
14.3	**FlexCAN eDMA Driver**	**248**
14.3.1	Overview	248
14.3.2	Data Structure Documentation	248
14.3.3	Typedef Documentation	249
14.3.4	Function Documentation	249

Chapter GPIO: General-Purpose Input/Output Driver

15.1	**Overview**	**253**
15.2	**Data Structure Documentation**	**253**
15.2.1	struct gpio_pin_config_t	253
15.3	**Macro Definition Documentation**	**254**
15.3.1	FSL_GPIO_DRIVER_VERSION	254
15.4	**Enumeration Type Documentation**	**254**
15.4.1	gpio_pin_direction_t	254
15.5	**GPIO Driver**	**255**
15.5.1	Overview	255
15.5.2	Typical use case	255
15.5.3	Function Documentation	256

Contents

Section Number	Title	Page Number
15.6	**FGPIO Driver**	**259**
15.6.1	Overview	259
15.6.2	Typical use case	259
15.6.3	Function Documentation	260
Chapter	**I2C: Inter-Integrated Circuit Driver**	
16.1	**Overview**	**265**
16.2	**I2C Driver**	**266**
16.2.1	Overview	266
16.2.2	Typical use case	266
16.2.3	Data Structure Documentation	273
16.2.4	Macro Definition Documentation	278
16.2.5	Typedef Documentation	278
16.2.6	Enumeration Type Documentation	278
16.2.7	Function Documentation	280
16.3	**I2C eDMA Driver**	**295**
16.3.1	Overview	295
16.3.2	Data Structure Documentation	295
16.3.3	Typedef Documentation	296
16.3.4	Function Documentation	296
16.4	**I2C DMA Driver**	**299**
16.4.1	Overview	299
16.4.2	Data Structure Documentation	299
16.4.3	Typedef Documentation	300
16.4.4	Function Documentation	300
16.5	**I2C FreeRTOS Driver**	**302**
16.5.1	Overview	302
16.5.2	Function Documentation	302
Chapter	**LIN: Local Interconnect Network Driver**	
17.1	**Overview**	**305**
17.2	**LIN Driver**	**306**
17.2.1	Overview	306
17.2.2	Typical use case	306
17.2.3	Data Structure Documentation	313
17.2.4	Enumeration Type Documentation	318
17.2.5	Function Documentation	320
17.2.6	Variable Documentation	332

Contents

Section Number	Title	Page Number
17.3	**LIN LPUART Driver**	**333**
17.3.1	Overview	333
17.3.2	Enumeration Type Documentation	335
17.3.3	Function Documentation	337
Chapter	**LLWU: Low-Leakage Wakeup Unit Driver**	
18.1	**Overview**	**345**
18.2	**External wakeup pins configurations**	**345**
18.3	**Internal wakeup modules configurations**	**345**
18.4	**Digital pin filter for external wakeup pin configurations**	**345**
18.5	**Data Structure Documentation**	**346**
18.5.1	struct llwu_external_pin_filter_mode_t	346
18.6	**Macro Definition Documentation**	**346**
18.6.1	FSL_LLWU_DRIVER_VERSION	346
18.7	**Enumeration Type Documentation**	**346**
18.7.1	llwu_external_pin_mode_t	346
18.7.2	llwu_pin_filter_mode_t	347
18.8	**Function Documentation**	**347**
18.8.1	LLWU_SetExternalWakeupPinMode	347
18.8.2	LLWU_GetExternalWakeupPinFlag	347
18.8.3	LLWU_ClearExternalWakeupPinFlag	348
18.8.4	LLWU_EnableInternalModuleInterruptWakup	349
18.8.5	LLWU_GetInternalWakeupModuleFlag	349
18.8.6	LLWU_SetPinFilterMode	349
18.8.7	LLWU_GetPinFilterFlag	350
18.8.8	LLWU_ClearPinFilterFlag	350
Chapter	**LPTMR: Low-Power Timer**	
19.1	**Overview**	**351**
19.2	**Function groups**	**351**
19.2.1	Initialization and deinitialization	351
19.2.2	Timer period Operations	351
19.2.3	Start and Stop timer operations	351
19.2.4	Status	352
19.2.5	Interrupt	352

Contents

Section Number	Title	Page Number
19.3	**Typical use case**	**352**
19.3.1	LPTMR tick example	352
19.4	**Data Structure Documentation**	**354**
19.4.1	struct lptmr_config_t	354
19.5	**Enumeration Type Documentation**	**355**
19.5.1	lptmr_pin_select_t	355
19.5.2	lptmr_pin_polarity_t	355
19.5.3	lptmr_timer_mode_t	355
19.5.4	lptmr_prescaler_glitch_value_t	356
19.5.5	lptmr_prescaler_clock_select_t	356
19.5.6	lptmr_interrupt_enable_t	356
19.5.7	lptmr_status_flags_t	357
19.6	**Function Documentation**	**357**
19.6.1	LPTMR_Init	357
19.6.2	LPTMR_Deinit	357
19.6.3	LPTMR_GetDefaultConfig	357
19.6.4	LPTMR_EnableInterrupts	358
19.6.5	LPTMR_DisableInterrupts	358
19.6.6	LPTMR_GetEnabledInterrupts	358
19.6.7	LPTMR_GetStatusFlags	358
19.6.8	LPTMR_ClearStatusFlags	359
19.6.9	LPTMR_SetTimerPeriod	359
19.6.10	LPTMR_GetCurrentTimerCount	359
19.6.11	LPTMR_StartTimer	360
19.6.12	LPTMR_StopTimer	360
Chapter	**LPUART: Low Power UART Driver**	
20.1	**Overview**	**361**
20.2	**LPUART Driver**	**362**
20.2.1	Overview	362
20.2.2	Typical use case	362
20.2.3	Data Structure Documentation	366
20.2.4	Macro Definition Documentation	368
20.2.5	Typedef Documentation	368
20.2.6	Enumeration Type Documentation	368
20.2.7	Function Documentation	371
20.3	**LPUART DMA Driver**	**386**
20.3.1	Overview	386
20.3.2	Data Structure Documentation	386

Contents

Section Number	Title	Page Number
20.3.3	Typedef Documentation	387
20.3.4	Function Documentation	387
20.4	**LPUART eDMA Driver**	**391**
20.4.1	Overview	391
20.4.2	Data Structure Documentation	392
20.4.3	Typedef Documentation	393
20.4.4	Function Documentation	393
20.5	**LPUART FreeRTOS Driver**	**397**
20.5.1	Overview	397
20.5.2	Data Structure Documentation	397
20.5.3	Function Documentation	398
Chapter	**LTC: LP Trusted Cryptography**	
21.1	**Overview**	**401**
21.2	**LTC Driver Initialization and Configuration**	**401**
21.3	**Comments about API usage in RTOS**	**401**
21.4	**Comments about API usage in interrupt handler**	**401**
21.5	**LTC Driver Examples**	**402**
21.5.1	Simple examples	402
21.6	**Macro Definition Documentation**	**404**
21.6.1	FSL_LTC_DRIVER_VERSION	404
21.7	**Function Documentation**	**404**
21.7.1	LTC_Init	404
21.7.2	LTC_Deinit	404
21.8	**LTC Blocking APIs**	**406**
21.8.1	Overview	406
21.8.2	LTC DES driver	407
21.8.3	LTC AES driver	424
21.8.4	LTC HASH driver	431
21.8.5	LTC PKHA driver	435
21.9	**LTC Non-blocking eDMA APIs**	**451**
21.9.1	Overview	451
21.9.2	Data Structure Documentation	451
21.9.3	Typedef Documentation	453
21.9.4	Function Documentation	454

Contents

Section Number	Title	Page Number
21.9.5	LTC eDMA DES driver	456
21.9.6	LTC eDMA AES driver	476
Chapter	**PIT: Periodic Interrupt Timer**	
22.1	**Overview**	**483**
22.2	**Function groups**	**483**
22.2.1	Initialization and deinitialization	483
22.2.2	Timer period Operations	483
22.2.3	Start and Stop timer operations	483
22.2.4	Status	484
22.2.5	Interrupt	484
22.3	**Typical use case**	**484**
22.3.1	PIT tick example	484
22.4	**Data Structure Documentation**	**486**
22.4.1	struct pit_config_t	486
22.5	**Enumeration Type Documentation**	**486**
22.5.1	pit_chnl_t	486
22.5.2	pit_interrupt_enable_t	487
22.5.3	pit_status_flags_t	487
22.6	**Function Documentation**	**487**
22.6.1	PIT_Init	487
22.6.2	PIT_Deinit	487
22.6.3	PIT_GetDefaultConfig	487
22.6.4	PIT_SetTimerChainMode	488
22.6.5	PIT_EnableInterrupts	488
22.6.6	PIT_DisableInterrupts	488
22.6.7	PIT_GetEnabledInterrupts	489
22.6.8	PIT_GetStatusFlags	489
22.6.9	PIT_ClearStatusFlags	489
22.6.10	PIT_SetTimerPeriod	490
22.6.11	PIT_GetCurrentTimerCount	490
22.6.12	PIT_StartTimer	491
22.6.13	PIT_StopTimer	491
22.6.14	PIT_GetLifetimeTimerCount	491
Chapter	**PMC: Power Management Controller**	
23.1	**Overview**	**493**
23.2	**Data Structure Documentation**	**494**

Contents

Section Number	Title	Page Number
23.2.1	struct pmc_low_volt_detect_config_t	494
23.2.2	struct pmc_low_volt_warning_config_t	494
23.2.3	struct pmc_bandgap_buffer_config_t	495
23.3	**Macro Definition Documentation**	**495**
23.3.1	FSL_PMC_DRIVER_VERSION	495
23.4	**Enumeration Type Documentation**	**495**
23.4.1	pmc_low_volt_detect_volt_select_t	495
23.4.2	pmc_low_volt_warning_volt_select_t	495
23.4.3	pmc_vlp_freq_mode_t	495
23.5	**Function Documentation**	**496**
23.5.1	PMC_ConfigureLowVoltDetect	496
23.5.2	PMC_GetLowVoltDetectFlag	497
23.5.3	PMC_ClearLowVoltDetectFlag	497
23.5.4	PMC_ConfigureLowVoltWarning	497
23.5.5	PMC_GetLowVoltWarningFlag	498
23.5.6	PMC_ClearLowVoltWarningFlag	498
23.5.7	PMC_ConfigureBandgapBuffer	498
23.5.8	PMC_GetPeriphIOIsolationFlag	499
23.5.9	PMC_ClearPeriphIOIsolationFlag	499
23.5.10	PMC_IsRegulatorInRunRegulation	499
Chapter	**PORT: Port Control and Interrupts**	
24.1	**Overview**	**501**
24.2	**Typical configuration use case**	**501**
24.2.1	Input PORT configuration	501
24.2.2	I2C PORT Configuration	501
24.3	**Data Structure Documentation**	**503**
24.3.1	struct port_pin_config_t	503
24.4	**Macro Definition Documentation**	**503**
24.4.1	FSL_PORT_DRIVER_VERSION	503
24.5	**Enumeration Type Documentation**	**503**
24.5.1	_port_pull	503
24.5.2	_port_slew_rate	504
24.5.3	_port_passive_filter_enable	504
24.5.4	_port_drive_strength	504
24.5.5	port_mux_t	504
24.5.6	port_interrupt_t	505

Contents

Section Number	Title	Page Number
24.6	**Function Documentation**	**505**
24.6.1	PORT_SetPinConfig	505
24.6.2	PORT_SetMultiplePinsConfig	506
24.6.3	PORT_SetPinMux	507
24.6.4	PORT_SetPinInterruptConfig	508
24.6.5	PORT_SetPinDriveStrength	508
24.6.6	PORT_GetPinsInterruptFlags	509
24.6.7	PORT_ClearPinsInterruptFlags	509
Chapter	**RCM: Reset Control Module Driver**	
25.1	**Overview**	**511**
25.2	**Data Structure Documentation**	**512**
25.2.1	struct rcm_reset_pin_filter_config_t	512
25.3	**Macro Definition Documentation**	**512**
25.3.1	FSL_RCM_DRIVER_VERSION	512
25.4	**Enumeration Type Documentation**	**512**
25.4.1	rcm_reset_source_t	512
25.4.2	rcm_run_wait_filter_mode_t	512
25.5	**Function Documentation**	**513**
25.5.1	RCM_GetPreviousResetSources	513
25.5.2	RCM_ConfigureResetPinFilter	513
Chapter	**RTC: Real Time Clock**	
26.1	**Overview**	**515**
26.2	**Function groups**	**515**
26.2.1	Initialization and deinitialization	515
26.2.2	Set & Get Datetime	515
26.2.3	Set & Get Alarm	515
26.2.4	Start & Stop timer	516
26.2.5	Status	516
26.2.6	Interrupt	516
26.2.7	RTC Oscillator	516
26.2.8	Monotonic Counter	516
26.3	**Typical use case**	**516**
26.3.1	RTC tick example	516
26.4	**Data Structure Documentation**	**519**
26.4.1	struct rtc_datetime_t	519

Contents

Section Number	Title	Page Number
26.4.2	struct rtc_config_t	520
26.5	**Enumeration Type Documentation**	**521**
26.5.1	rtc_interrupt_enable_t	521
26.5.2	rtc_status_flags_t	521
26.5.3	rtc_osc_cap_load_t	521
26.6	**Function Documentation**	**521**
26.6.1	RTC_Init	521
26.6.2	RTC_Deinit	522
26.6.3	RTC_GetDefaultConfig	522
26.6.4	RTC_SetDatetime	522
26.6.5	RTC_GetDatetime	523
26.6.6	RTC_SetAlarm	523
26.6.7	RTC_GetAlarm	523
26.6.8	RTC_EnableInterrupts	524
26.6.9	RTC_DisableInterrupts	524
26.6.10	RTC_GetEnabledInterrupts	524
26.6.11	RTC_GetStatusFlags	524
26.6.12	RTC_ClearStatusFlags	525
26.6.13	RTC_SetClockSource	525
26.6.14	RTC_StartTimer	525
26.6.15	RTC_StopTimer	526
26.6.16	RTC_SetOscCapLoad	526
26.6.17	RTC_Reset	526
Chapter	**SIM: System Integration Module Driver**	
27.1	**Overview**	**527**
27.2	**Data Structure Documentation**	**527**
27.2.1	struct sim_uid_t	527
27.3	**Enumeration Type Documentation**	**528**
27.3.1	_sim_flash_mode	528
27.4	**Function Documentation**	**528**
27.4.1	SIM_GetUniqueId	528
27.4.2	SIM_SetFlashMode	528
Chapter	**SMC: System Mode Controller Driver**	
28.1	**Overview**	**529**
28.2	**Typical use case**	**529**
28.2.1	Enter wait or stop modes	529

Contents

Section Number	Title	Page Number
28.3	**Data Structure Documentation**	**531**
28.3.1	struct smc_power_mode_lls_config_t	531
28.3.2	struct smc_power_mode_vlls_config_t	531
28.4	**Macro Definition Documentation**	**532**
28.4.1	FSL_SMC_DRIVER_VERSION	532
28.5	**Enumeration Type Documentation**	**532**
28.5.1	smc_power_mode_protection_t	532
28.5.2	smc_power_state_t	532
28.5.3	smc_run_mode_t	532
28.5.4	smc_stop_mode_t	532
28.5.5	smc_stop_submode_t	533
28.5.6	smc_partial_stop_option_t	533
28.5.7	_smc_status	533
28.6	**Function Documentation**	**533**
28.6.1	SMC_SetPowerModeProtection	533
28.6.2	SMC_GetPowerModeState	534
28.6.3	SMC_PreEnterStopModes	534
28.6.4	SMC_PostExitStopModes	534
28.6.5	SMC_PreEnterWaitModes	534
28.6.6	SMC_PostExitWaitModes	534
28.6.7	SMC_SetPowerModeRun	535
28.6.8	SMC_SetPowerModeWait	536
28.6.9	SMC_SetPowerModeStop	536
28.6.10	SMC_SetPowerModeVlpr	536
28.6.11	SMC_SetPowerModeVlpw	537
28.6.12	SMC_SetPowerModeVlps	537
28.6.13	SMC_SetPowerModeLls	537
28.6.14	SMC_SetPowerModeVlls	538
Chapter	**TPM: Timer PWM Module**	
29.1	**Overview**	**539**
29.2	**Typical use case**	**540**
29.2.1	PWM output	540
29.3	**Data Structure Documentation**	**544**
29.3.1	struct tpm_chnl_pwm_signal_param_t	544
29.3.2	struct tpm_dual_edge_capture_param_t	545
29.3.3	struct tpm_phase_params_t	545
29.3.4	struct tpm_config_t	546
29.4	**Enumeration Type Documentation**	**546**

Contents

Section Number	Title	Page Number
29.4.1	tpm_chnl_t	546
29.4.2	tpm_pwm_mode_t	547
29.4.3	tpm_pwm_level_select_t	547
29.4.4	tpm_trigger_select_t	547
29.4.5	tpm_trigger_source_t	547
29.4.6	tpm_output_compare_mode_t	548
29.4.7	tpm_input_capture_edge_t	548
29.4.8	tpm_quad_decode_mode_t	548
29.4.9	tpm_phase_polarity_t	548
29.4.10	tpm_clock_source_t	548
29.4.11	tpm_clock_prescale_t	549
29.4.12	tpm_interrupt_enable_t	549
29.4.13	tpm_status_flags_t	549
29.5	**Function Documentation**	**549**
29.5.1	TPM_Init	549
29.5.2	TPM_Deinit	550
29.5.3	TPM_GetDefaultConfig	550
29.5.4	TPM_SetupPwm	550
29.5.5	TPM_UpdatePwmDutycycle	551
29.5.6	TPM_UpdateChnlEdgeLevelSelect	551
29.5.7	TPM_SetupInputCapture	552
29.5.8	TPM_SetupOutputCompare	552
29.5.9	TPM_SetupDualEdgeCapture	552
29.5.10	TPM_SetupQuadDecode	553
29.5.11	TPM_EnableInterrupts	553
29.5.12	TPM_DisableInterrupts	553
29.5.13	TPM_GetEnabledInterrupts	554
29.5.14	TPM_GetStatusFlags	554
29.5.15	TPM_ClearStatusFlags	554
29.5.16	TPM_SetTimerPeriod	555
29.5.17	TPM_GetCurrentTimerCount	555
29.5.18	TPM_StartTimer	555
29.5.19	TPM_StopTimer	556
Chapter	**TRNG: True Random Number Generator**	
30.1	**Overview**	**557**
30.2	**TRNG Initialization**	**557**
30.3	**Get random data from TRNG**	**557**
30.4	**Data Structure Documentation**	**559**
30.4.1	struct trng_statistical_check_limit_t	559

Contents

Section Number	Title	Page Number
30.4.2	struct trng_config_t	559
30.5	**Macro Definition Documentation**	**561**
30.5.1	FSL_TRNG_DRIVER_VERSION	561
30.6	**Enumeration Type Documentation**	**561**
30.6.1	trng_sample_mode_t	561
30.6.2	trng_clock_mode_t	562
30.6.3	trng_ring_osc_div_t	562
30.7	**Function Documentation**	**562**
30.7.1	TRNG_GetDefaultConfig	562
30.7.2	TRNG_Init	563
30.7.3	TRNG_Deinit	563
30.7.4	TRNG_GetRandomData	563
Chapter	**VREF: Voltage Reference Driver**	
31.1	**Overview**	**565**
31.2	**Typical use case and example**	**565**
31.3	**Data Structure Documentation**	**566**
31.3.1	struct vref_config_t	566
31.4	**Macro Definition Documentation**	**566**
31.4.1	FSL_VREF_DRIVER_VERSION	566
31.5	**Enumeration Type Documentation**	**566**
31.5.1	vref_buffer_mode_t	566
31.6	**Function Documentation**	**566**
31.6.1	VREF_Init	566
31.6.2	VREF_Deinit	567
31.6.3	VREF_GetDefaultConfig	567
31.6.4	VREF_SetTrimVal	567
31.6.5	VREF_GetTrimVal	568
Chapter	**Clock Driver**	
32.1	**Overview**	**569**
32.2	**Get frequency**	**569**
32.3	**External clock frequency**	**569**
32.4	**Data Structure Documentation**	**575**

Contents

Section Number	Title	Page Number
32.4.1	struct sim_clock_config_t	575
32.4.2	struct osc_config_t	576
32.4.3	struct mcg_config_t	576
32.5	**Macro Definition Documentation**	**577**
32.5.1	MCG_CONFIG_CHECK_PARAM	577
32.5.2	FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL	578
32.5.3	FSL_CLOCK_DRIVER_VERSION	578
32.5.4	DMAMUX_CLOCKS	578
32.5.5	RTC_CLOCKS	578
32.5.6	PIT_CLOCKS	578
32.5.7	PORT_CLOCKS	578
32.5.8	DSPI_CLOCKS	579
32.5.9	LPUART_CLOCKS	579
32.5.10	FLEXCAN_CLOCKS	579
32.5.11	LPTMR_CLOCKS	579
32.5.12	ADC16_CLOCKS	579
32.5.13	TRNG_CLOCKS	580
32.5.14	EDMA_CLOCKS	580
32.5.15	CMT_CLOCKS	580
32.5.16	TPM_CLOCKS	580
32.5.17	LTC_CLOCKS	580
32.5.18	I2C_CLOCKS	581
32.5.19	CMP_CLOCKS	581
32.5.20	VREF_CLOCKS	581
32.5.21	DCDC_CLOCKS	581
32.5.22	SYS_CLK	581
32.6	**Enumeration Type Documentation**	**581**
32.6.1	clock_name_t	581
32.6.2	clock_ip_name_t	582
32.6.3	osc_mode_t	582
32.6.4	mcg_fll_src_t	582
32.6.5	mcg_irc_mode_t	582
32.6.6	mcg_dmx32_t	582
32.6.7	mcg_drs_t	583
32.6.8	mcg_pll_ref_src_t	583
32.6.9	mcg_clkout_src_t	583
32.6.10	mcg_atm_select_t	583
32.6.11	mcg_oscsel_t	583
32.6.12	mcg_pll_clk_select_t	584
32.6.13	mcg_monitor_mode_t	584
32.6.14	_mcg_status	584
32.6.15	_mcg_status_flags_t	584
32.6.16	_mcg_irclk_enable_mode	584

Contents

Section Number	Title	Page Number
32.6.17	mcg_mode_t	585
32.7	**Function Documentation**	**585**
32.7.1	CLOCK_EnableClock	585
32.7.2	CLOCK_DisableClock	585
32.7.3	CLOCK_isEnabledClock	585
32.7.4	CLOCK_SetEr32kClock	585
32.7.5	CLOCK_SetLpuart0Clock	586
32.7.6	CLOCK_SetLpuart1Clock	586
32.7.7	CLOCK_SetTpmClock	586
32.7.8	CLOCK_SetClkOutClock	586
32.7.9	CLOCK_SetOutDiv	586
32.7.10	CLOCK_GetFreq	587
32.7.11	CLOCK_GetCoreSysClkFreq	587
32.7.12	CLOCK_GetPlatClkFreq	587
32.7.13	CLOCK_GetBusClkFreq	587
32.7.14	CLOCK_GetFlashClkFreq	588
32.7.15	CLOCK_GetEr32kClkFreq	588
32.7.16	CLOCK_GetOsc0ErClkFreq	588
32.7.17	CLOCK_SetSimConfig	588
32.7.18	CLOCK_SetSimSafeDivs	588
32.7.19	CLOCK_GetOutClkFreq	589
32.7.20	CLOCK_GetFllFreq	589
32.7.21	CLOCK_GetInternalRefClkFreq	589
32.7.22	CLOCK_GetFixedFreqClkFreq	589
32.7.23	CLOCK_SetLowPowerEnable	590
32.7.24	CLOCK_SetInternalRefClkConfig	590
32.7.25	CLOCK_SetExternalRefClkConfig	590
32.7.26	CLOCK_SetFllExtRefDiv	591
32.7.27	CLOCK_SetRtcOscMonitorMode	591
32.7.28	CLOCK_GetStatusFlags	591
32.7.29	CLOCK_ClearStatusFlags	592
32.7.30	CLOCK_InitOsc0	592
32.7.31	CLOCK_DeinitOsc0	592
32.7.32	CLOCK_SetXtal0Freq	592
32.7.33	CLOCK_SetXtal32Freq	593
32.7.34	CLOCK_TrimInternalRefClk	593
32.7.35	CLOCK_GetMode	594
32.7.36	CLOCK_SetFeiMode	594
32.7.37	CLOCK_SetFeeMode	594
32.7.38	CLOCK_SetFbiMode	595
32.7.39	CLOCK_SetFbeMode	596
32.7.40	CLOCK_SetBlpiMode	597
32.7.41	CLOCK_SetBlpeMode	597
32.7.42	CLOCK_ExternalModeToFbeModeQuick	598

Contents

Section Number	Title	Page Number
32.7.43	CLOCK_InternalModeToFbiModeQuick	598
32.7.44	CLOCK_BootToFeiMode	598
32.7.45	CLOCK_BootToFeeMode	599
32.7.46	CLOCK_BootToBlpiMode	600
32.7.47	CLOCK_BootToBlpeMode	601
32.7.48	CLOCK_SetMcgConfig	601
32.8	**Variable Documentation**	**602**
32.8.1	g_xtal0Freq	602
32.8.2	g_xtal32Freq	602
32.9	**Multipurpose Clock Generator (MCG)**	**603**
32.9.1	Function description	603
32.9.2	Typical use case	605
32.9.3	Code Configuration Option	608
Chapter	**DMA Manager**	
33.1	**Overview**	**611**
33.2	**Function groups**	**611**
33.2.1	DMAMGR Initialization and De-initialization	611
33.2.2	DMAMGR Operation	611
33.3	**Typical use case**	**611**
33.3.1	DMAMGR static channel allocattion	611
33.3.2	DMAMGR dynamic channel allocation	611
33.4	**Data Structure Documentation**	**612**
33.4.1	struct dmamanager_handle_t	612
33.5	**Macro Definition Documentation**	**613**
33.5.1	DMAMGR_DYNAMIC_ALLOCATE	613
33.6	**Enumeration Type Documentation**	**613**
33.6.1	_dma_manager_status	613
33.7	**Function Documentation**	**613**
33.7.1	DMAMGR_Init	613
33.7.2	DMAMGR_Deinit	614
33.7.3	DMAMGR_RequestChannel	614
33.7.4	DMAMGR_ReleaseChannel	615
33.7.5	DMAMGR_IsChannelOccupied	616

Contents

Section Number	Title	Page Number
Chapter	**Secure Digital Card/Embedded MultiMedia Card (CARD)**	
34.1	**Overview**	**617**
34.2	**Data Structure Documentation**	**622**
34.2.1	struct sd_card_t	622
34.2.2	struct sdio_card_t	623
34.2.3	struct mmc_card_t	624
34.2.4	struct mmc_boot_config_t	625
34.3	**Macro Definition Documentation**	**625**
34.3.1	FSL_SDMMC_DRIVER_VERSION	625
34.4	**Enumeration Type Documentation**	**625**
34.4.1	_sdmmc_status	625
34.4.2	_sd_card_flag	626
34.4.3	_mmc_card_flag	627
34.4.4	card_operation_voltage_t	627
34.4.5	_host_endian_mode	627
34.5	**Function Documentation**	**628**
34.5.1	SD_Init	628
34.5.2	SD_Deinit	629
34.5.3	SD_CheckReadOnly	630
34.5.4	SD_ReadBlocks	630
34.5.5	SD_WriteBlocks	631
34.5.6	SD_EraseBlocks	632
34.5.7	MMC_Init	632
34.5.8	MMC_Deinit	633
34.5.9	MMC_CheckReadOnly	633
34.5.10	MMC_ReadBlocks	634
34.5.11	MMC_WriteBlocks	634
34.5.12	MMC_EraseGroups	635
34.5.13	MMC_SelectPartition	636
34.5.14	MMC_SetBootConfig	636
34.5.15	SDIO_CardInActive	637
34.5.16	SDIO_IO_Write_Direct	637
34.5.17	SDIO_IO_Read_Direct	638
34.5.18	SDIO_IO_Write_Extended	639
34.5.19	SDIO_IO_Read_Extended	639
34.5.20	SDIO_GetCardCapability	640
34.5.21	SDIO_SetBlockSize	640
34.5.22	SDIO_CardReset	641
34.5.23	SDIO_SetDataBusWidth	641
34.5.24	SDIO_SwitchToHighSpeed	642

Contents

Section Number	Title	Page Number
34.5.25	SDIO_ReadCIS	642
34.5.26	SDIO_Init	643
34.5.27	SDIO_EnableIOInterrupt	644
34.5.28	SDIO_EnableIO	645
34.5.29	SDIO_SelectIO	645
34.5.30	SDIO_AbortIO	645
34.5.31	SDIO_DeInit	646
34.5.32	HOST_NotSupport	646
34.5.33	CardInsertDetect	646
34.5.34	HOST_Init	647
34.5.35	HOST_Reset	647
34.5.36	HOST_Deinit	647
Chapter	**SPI based Secure Digital Card (SDSPI)**	
35.1	**Overview**	**649**
35.2	**Data Structure Documentation**	**651**
35.2.1	struct sdspi_command_t	651
35.2.2	struct sdspi_host_t	651
35.2.3	struct sdspi_card_t	651
35.3	**Enumeration Type Documentation**	**652**
35.3.1	_sdspi_status	652
35.3.2	_sdspi_card_flag	653
35.3.3	sdspi_response_type_t	653
35.4	**Function Documentation**	**653**
35.4.1	SDSPI_Init	653
35.4.2	SDSPI_Deinit	654
35.4.3	SDSPI_CheckReadOnly	654
35.4.4	SDSPI_ReadBlocks	655
35.4.5	SDSPI_WriteBlocks	655
Chapter	**Debug Console**	
36.1	**Overview**	**657**
36.2	**Function groups**	**657**
36.2.1	Initialization	657
36.2.2	Advanced Feature	658
36.3	**Typical use case**	**661**
36.4	**Semihosting**	**663**
36.4.1	Guide Semihosting for IAR	663

Contents

Section Number	Title	Page Number
36.4.2	Guide Semihosting for Keil µVision	663
36.4.3	Guide Semihosting for KDS	665
36.4.4	Guide Semihosting for ATL	665
36.4.5	Guide Semihosting for ARMGCC	666
Chapter	**Notification Framework**	
37.1	**Overview**	**669**
37.2	**Notifier Overview**	**669**
37.3	**Data Structure Documentation**	**671**
37.3.1	struct notifier_notification_block_t	671
37.3.2	struct notifier_callback_config_t	672
37.3.3	struct notifier_handle_t	672
37.4	**Typedef Documentation**	**673**
37.4.1	notifier_user_config_t	673
37.4.2	notifier_user_function_t	673
37.4.3	notifier_callback_t	674
37.5	**Enumeration Type Documentation**	**674**
37.5.1	_notifier_status	674
37.5.2	notifier_policy_t	675
37.5.3	notifier_notification_type_t	675
37.5.4	notifier_callback_type_t	675
37.6	**Function Documentation**	**676**
37.6.1	NOTIFIER_CreateHandle	676
37.6.2	NOTIFIER_SwitchConfig	677
37.6.3	NOTIFIER_GetErrorCallbackIndex	678
Chapter	**Shell**	
38.1	**Overview**	**679**
38.2	**Function groups**	**679**
38.2.1	Initialization	679
38.2.2	Advanced Feature	679
38.2.3	Shell Operation	680
38.3	**Data Structure Documentation**	**681**
38.3.1	struct shell_context_struct	681
38.3.2	struct shell_command_context_t	682
38.3.3	struct shell_command_context_list_t	682

Contents

Section Number	Title	Page Number
38.4	**Macro Definition Documentation**	**683**
38.4.1	SHELL_USE_HISTORY	683
38.4.2	SHELL_SEARCH_IN_HIST	683
38.4.3	SHELL_USE_FILE_STREAM	683
38.4.4	SHELL_AUTO_COMPLETE	683
38.4.5	SHELL_BUFFER_SIZE	683
38.4.6	SHELL_MAX_ARGS	683
38.4.7	SHELL_HIST_MAX	683
38.4.8	SHELL_MAX_CMD	683
38.5	**Typedef Documentation**	**683**
38.5.1	send_data_cb_t	683
38.5.2	recv_data_cb_t	683
38.5.3	printf_data_t	683
38.5.4	cmd_function_t	683
38.6	**Enumeration Type Documentation**	**683**
38.6.1	fun_key_status_t	683
38.7	**Function Documentation**	**684**
38.7.1	SHELL_Init	684
38.7.2	SHELL_RegisterCommand	684
38.7.3	SHELL_Main	684

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, USB stack, 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 KEx Web UI 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 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:
  - A USB device, host, and OTG stack with comprehensive USB class support.
  - 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
  - Keil MDK
  - MCUXpresso IDE

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 `kex.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

Table 2: MCUXpresso SDK Folder Structure

Chapter 2
Driver errors status

- kStatus_DSPI_Error = 601
- kStatus_EDMA_QueueFull = 5100
- kStatus_EDMA_Busy = 5101
- kStatus_SMC_StopAbort = 3900
- kStatus_DMAMGR_ChannelOccupied = 5200
- kStatus_DMAMGR_ChannelNotUsed = 5201
- kStatus_DMAMGR_NoFreeChannel = 5202
- kStatus_NOTIFIER_ErrorNotificationBefore = 9800
- kStatus_NOTIFIER_ErrorNotificationAfter = 9801

Chapter 3
Architectural Overview

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

Overview

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

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

Figure 1: MCUXpresso SDK Block Diagram

MCU header files

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

mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers. The overall SoC header file provides a 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/DMA driver/eDMA driver to quickly enable the peripherals and perform transfers.

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

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

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

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

Interrupt handling for transactional APIs

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

```
        PUBWEAK SPI0_IRQHandler
        PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```

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

The first level of the weak implementation are the functions defined in the vector table. In the devices/<-DEVICE_NAME>/<TOOLCHAIN>/startup_<DEVICE_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0_DriverIRQHandler) jumps to itself (B .). The MCUXpresso SDK drivers with transactional APIs provide the reimplementation of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0_-DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

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

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

Feature Header Files

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

Application

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

Chapter 4
Trademarks

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

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

NXP reserves the right to make changes without further notice to any products herein. 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/Sales-TermsandConditions

NXP, the NXP logo, Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of NXP B.V. Tower is a trademark of NXP B.V. All other product or service names are the property of their respective owners. ARM, ARM powered logo, Keil, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Chapter 5
ADC16: 16-bit SAR Analog-to-Digital Converter Driver

5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 16-bit SAR Analog-to-Digital Converter (ADC16) module of MCUXpresso SDK devices.

5.2 Typical use case

5.2.1 Polling Configuration

```
    adc16_config_t adc16ConfigStruct;
    adc16_channel_config_t adc16ChannelConfigStruct;

    ADC16_Init(DEMO_ADC16_INSTANCE);
    ADC16_GetDefaultConfig(&adc16ConfigStruct);
    ADC16_Configure(DEMO_ADC16_INSTANCE, &adc16ConfigStruct);
    ADC16_EnableHardwareTrigger(DEMO_ADC16_INSTANCE, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_Success == ADC16_DoAutoCalibration(DEMO_ADC16_INSTANCE))
    {
        PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
    }
    else
    {
        PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
    }
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION

    adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
    adc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
      false;
#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) && FSL_FEATURE_ADC16_HAS_DIFF_MODE
    adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif // FSL_FEATURE_ADC16_HAS_DIFF_MODE

    while(1)
    {
        GETCHAR(); // Input any key in terminal console.
        ADC16_ChannelConfigure(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
        while (kADC16_ChannelConversionDoneFlag !=
      ADC16_ChannelGetStatusFlags(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP))
        {
        }
        PRINTF("ADC Value: %d\r\n", ADC16_ChannelGetConversionValue(DEMO_ADC16_INSTANCE,
      DEMO_ADC16_CHANNEL_GROUP));
    }
```

5.2.2 Interrupt Configuration

```
    volatile bool g_Adc16ConversionDoneFlag = false;
    volatile uint32_t g_Adc16ConversionValue;
    volatile uint32_t g_Adc16InterruptCount = 0U;
```

```
    // ...

    adc16_config_t adc16ConfigStruct;
    adc16_channel_config_t adc16ChannelConfigStruct;

    ADC16_Init(DEMO_ADC16_INSTANCE);
    ADC16_GetDefaultConfig(&adc16ConfigStruct);
    ADC16_Configure(DEMO_ADC16_INSTANCE, &adc16ConfigStruct);
    ADC16_EnableHardwareTrigger(DEMO_ADC16_INSTANCE, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (ADC16_DoAutoCalibration(DEMO_ADC16_INSTANCE))
    {
        PRINTF("ADC16_DoAutoCalibration() Done.\r\n");
    }
    else
    {
        PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
    }
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION

    adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
    adc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
      true; // Enable the interrupt.
#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) && FSL_FEATURE_ADC16_HAS_DIFF_MODE
    adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif // FSL_FEATURE_ADC16_HAS_DIFF_MODE

    while(1)
    {
        GETCHAR(); // Input a key in the terminal console.
        g_Adc16ConversionDoneFlag = false;
        ADC16_ChannelConfigure(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP, &adc16ChannelConfigStruct);
        while (!g_Adc16ConversionDoneFlag)
        {
        }
        PRINTF("ADC Value: %d\r\n", g_Adc16ConversionValue);
        PRINTF("ADC Interrupt Count: %d\r\n", g_Adc16InterruptCount);
    }

    // ...

    void DEMO_ADC16_IRQHandler(void)
    {
        g_Adc16ConversionDoneFlag = true;
        // Read the conversion result to clear the conversion completed flag.
        g_Adc16ConversionValue = ADC16_ChannelConversionValue(DEMO_ADC16_INSTANCE, DEMO_ADC16_CHANNEL_GROUP
      );
        g_Adc16InterruptCount++;
    }
```

Data Structures

- struct adc16_config_t
  ADC16 converter configuration. More...
- struct adc16_hardware_compare_config_t
  ADC16 Hardware comparison configuration. More...
- struct adc16_channel_config_t
  ADC16 channel conversion configuration. More...

Enumerations

- enum _adc16_channel_status_flags { kADC16_ChannelConversionDoneFlag = ADC_SC1_COC-O_MASK }

Channel status flags.

- enum _adc16_status_flags {
  kADC16_ActiveFlag = ADC_SC2_ADACT_MASK,
  kADC16_CalibrationFailedFlag = ADC_SC3_CALF_MASK }
  Converter status flags.
- enum adc16_channel_mux_mode_t {
  kADC16_ChannelMuxA = 0U,
  kADC16_ChannelMuxB = 1U }
  Channel multiplexer mode for each channel.
- enum adc16_clock_divider_t {
  kADC16_ClockDivider1 = 0U,
  kADC16_ClockDivider2 = 1U,
  kADC16_ClockDivider4 = 2U,
  kADC16_ClockDivider8 = 3U }
  Clock divider for the converter.
- enum adc16_resolution_t {
  kADC16_Resolution8or9Bit = 0U,
  kADC16_Resolution12or13Bit = 1U,
  kADC16_Resolution10or11Bit = 2U,
  kADC16_ResolutionSE8Bit = kADC16_Resolution8or9Bit,
  kADC16_ResolutionSE12Bit = kADC16_Resolution12or13Bit,
  kADC16_ResolutionSE10Bit = kADC16_Resolution10or11Bit,
  kADC16_ResolutionDF9Bit = kADC16_Resolution8or9Bit,
  kADC16_ResolutionDF13Bit = kADC16_Resolution12or13Bit,
  kADC16_ResolutionDF11Bit = kADC16_Resolution10or11Bit,
  kADC16_Resolution16Bit = 3U,
  kADC16_ResolutionSE16Bit = kADC16_Resolution16Bit,
  kADC16_ResolutionDF16Bit = kADC16_Resolution16Bit }
  Converter's resolution.
- enum adc16_clock_source_t {
  kADC16_ClockSourceAlt0 = 0U,
  kADC16_ClockSourceAlt1 = 1U,
  kADC16_ClockSourceAlt2 = 2U,
  kADC16_ClockSourceAlt3 = 3U,
  kADC16_ClockSourceAsynchronousClock = kADC16_ClockSourceAlt3 }
  Clock source.
- enum adc16_long_sample_mode_t {
  kADC16_LongSampleCycle24 = 0U,
  kADC16_LongSampleCycle16 = 1U,
  kADC16_LongSampleCycle10 = 2U,
  kADC16_LongSampleCycle6 = 3U,
  kADC16_LongSampleDisabled = 4U }
  Long sample mode.
- enum adc16_reference_voltage_source_t {
  kADC16_ReferenceVoltageSourceVref = 0U,
  kADC16_ReferenceVoltageSourceValt = 1U }

Reference voltage source.

- enum adc16_hardware_average_mode_t {
  kADC16_HardwareAverageCount4 = 0U,
  kADC16_HardwareAverageCount8 = 1U,
  kADC16_HardwareAverageCount16 = 2U,
  kADC16_HardwareAverageCount32 = 3U,
  kADC16_HardwareAverageDisabled = 4U }
  Hardware average mode.
- enum adc16_hardware_compare_mode_t {
  kADC16_HardwareCompareMode0 = 0U,
  kADC16_HardwareCompareMode1 = 1U,
  kADC16_HardwareCompareMode2 = 2U,
  kADC16_HardwareCompareMode3 = 3U }
  Hardware compare mode.

Driver version

- #define FSL_ADC16_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  ADC16 driver version 2.0.0.

Initialization

- void ADC16_Init (ADC_Type *base, const adc16_config_t *config)
  Initializes the ADC16 module.
- void ADC16_Deinit (ADC_Type *base)
  De-initializes the ADC16 module.
- void ADC16_GetDefaultConfig (adc16_config_t *config)
  Gets an available pre-defined settings for the converter's configuration.
- status_t ADC16_DoAutoCalibration (ADC_Type *base)
  Automates the hardware calibration.
- static void ADC16_SetOffsetValue (ADC_Type *base, int16_t value)
  Sets the offset value for the conversion result.

Advanced Features

- static void ADC16_EnableDMA (ADC_Type *base, bool enable)
  Enables generating the DMA trigger when the conversion is complete.
- static void ADC16_EnableHardwareTrigger (ADC_Type *base, bool enable)
  Enables the hardware trigger mode.
- void ADC16_SetChannelMuxMode (ADC_Type *base, adc16_channel_mux_mode_t mode)
  Sets the channel mux mode.
- void ADC16_SetHardwareCompareConfig (ADC_Type *base, const adc16_hardware_compare_-config_t *config)
  Configures the hardware compare mode.
- void ADC16_SetHardwareAverage (ADC_Type *base, adc16_hardware_average_mode_t mode)
  Sets the hardware average mode.
- uint32_t ADC16_GetStatusFlags (ADC_Type *base)
  Gets the status flags of the converter.
- void ADC16_ClearStatusFlags (ADC_Type *base, uint32_t mask)
  Clears the status flags of the converter.

Conversion Channel

- void ADC16_SetChannelConfig (ADC_Type *base, uint32_t channelGroup, const adc16_channel_config_t *config)
  Configures the conversion channel.
- static uint32_t ADC16_GetChannelConversionValue (ADC_Type *base, uint32_t channelGroup)
  Gets the conversion value.
- uint32_t ADC16_GetChannelStatusFlags (ADC_Type *base, uint32_t channelGroup)
  Gets the status flags of channel.

5.3 Data Structure Documentation

5.3.1 struct adc16_config_t

Data Fields

- adc16_reference_voltage_source_t referenceVoltageSource
  Select the reference voltage source.
- adc16_clock_source_t clockSource
  Select the input clock source to converter.
- bool enableAsynchronousClock
  Enable the asynchronous clock output.
- adc16_clock_divider_t clockDivider
  Select the divider of input clock source.
- adc16_resolution_t resolution
  Select the sample resolution mode.
- adc16_long_sample_mode_t longSampleMode
  Select the long sample mode.
- bool enableHighSpeed
  Enable the high-speed mode.
- bool enableLowPower
  Enable low power.
- bool enableContinuousConversion
  Enable continuous conversion mode.

5.3.1.0.0.1 Field Documentation

5.3.1.0.0.1.1 adc16_reference_voltage_source_t adc16_config_t::referenceVoltageSource

5.3.1.0.0.1.2 adc16_clock_source_t adc16_config_t::clockSource

5.3.1.0.0.1.3 bool adc16_config_t::enableAsynchronousClock

5.3.1.0.0.1.4 adc16_clock_divider_t adc16_config_t::clockDivider

5.3.1.0.0.1.5 adc16_resolution_t adc16_config_t::resolution

5.3.1.0.0.1.6 adc16_long_sample_mode_t adc16_config_t::longSampleMode

5.3.1.0.0.1.7 bool adc16_config_t::enableHighSpeed

5.3.1.0.0.1.8 bool adc16_config_t::enableLowPower

5.3.1.0.0.1.9 bool adc16_config_t::enableContinuousConversion

5.3.2 struct adc16_hardware_compare_config_t

Data Fields

- adc16_hardware_compare_mode_t hardwareCompareMode
  Select the hardware compare mode.
- int16_t value1
  Setting value1 for hardware compare mode.
- int16_t value2
  Setting value2 for hardware compare mode.

5.3.2.0.0.2 Field Documentation

5.3.2.0.0.2.1 adc16_hardware_compare_mode_t adc16_hardware_compare_config_t::hardware-CompareMode

See "adc16_hardware_compare_mode_t".

5.3.2.0.0.2.2 int16_t adc16_hardware_compare_config_t::value1

5.3.2.0.0.2.3 int16_t adc16_hardware_compare_config_t::value2

5.3.3 struct adc16_channel_config_t

Data Fields

- uint32_t channelNumber
  Setting the conversion channel number.
- bool enableInterruptOnConversionCompleted

Generate an interrupt request once the conversion is completed.

- bool enableDifferentialConversion
  Using Differential sample mode.

5.3.3.0.0.3 Field Documentation

5.3.3.0.0.3.1 uint32_t adc16_channel_config_t::channelNumber

The available range is 0-31. See channel connection information for each chip in Reference Manual document.

5.3.3.0.0.3.2 bool adc16_channel_config_t::enableInterruptOnConversionCompleted

5.3.3.0.0.3.3 bool adc16_channel_config_t::enableDifferentialConversion

5.4 Macro Definition Documentation

5.4.1 #define FSL_ADC16_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

5.5 Enumeration Type Documentation

5.5.1 enum _adc16_channel_status_flags

Enumerator

kADC16_ChannelConversionDoneFlag Conversion done.

5.5.2 enum _adc16_status_flags

Enumerator

kADC16_ActiveFlag Converter is active.
kADC16_CalibrationFailedFlag Calibration is failed.

5.5.3 enum adc16_channel_mux_mode_t

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0_SE4a and ADC0_SE4b are the different channels that share the same channel number.

Enumerator

kADC16_ChannelMuxA For channel with channel mux a.
kADC16_ChannelMuxB For channel with channel mux b.

5.5.4 enum adc16_clock_divider_t

Enumerator

kADC16_ClockDivider1 For divider 1 from the input clock to the module.
kADC16_ClockDivider2 For divider 2 from the input clock to the module.
kADC16_ClockDivider4 For divider 4 from the input clock to the module.
kADC16_ClockDivider8 For divider 8 from the input clock to the module.

5.5.5 enum adc16_resolution_t

Enumerator

kADC16_Resolution8or9Bit Single End 8-bit or Differential Sample 9-bit.
kADC16_Resolution12or13Bit Single End 12-bit or Differential Sample 13-bit.
kADC16_Resolution10or11Bit Single End 10-bit or Differential Sample 11-bit.
kADC16_ResolutionSE8Bit Single End 8-bit.
kADC16_ResolutionSE12Bit Single End 12-bit.
kADC16_ResolutionSE10Bit Single End 10-bit.
kADC16_ResolutionDF9Bit Differential Sample 9-bit.
kADC16_ResolutionDF13Bit Differential Sample 13-bit.
kADC16_ResolutionDF11Bit Differential Sample 11-bit.
kADC16_Resolution16Bit Single End 16-bit or Differential Sample 16-bit.
kADC16_ResolutionSE16Bit Single End 16-bit.
kADC16_ResolutionDF16Bit Differential Sample 16-bit.

5.5.6 enum adc16_clock_source_t

Enumerator

kADC16_ClockSourceAlt0 Selection 0 of the clock source.
kADC16_ClockSourceAlt1 Selection 1 of the clock source.
kADC16_ClockSourceAlt2 Selection 2 of the clock source.
kADC16_ClockSourceAlt3 Selection 3 of the clock source.
kADC16_ClockSourceAsynchronousClock Using internal asynchronous clock.

5.5.7 enum adc16_long_sample_mode_t

Enumerator

kADC16_LongSampleCycle24 20 extra ADCK cycles, 24 ADCK cycles total.
kADC16_LongSampleCycle16 12 extra ADCK cycles, 16 ADCK cycles total.

kADC16_LongSampleCycle10 6 extra ADCK cycles, 10 ADCK cycles total.
kADC16_LongSampleCycle6 2 extra ADCK cycles, 6 ADCK cycles total.
kADC16_LongSampleDisabled Disable the long sample feature.

5.5.8 enum adc16_reference_voltage_source_t

Enumerator

kADC16_ReferenceVoltageSourceVref For external pins pair of VrefH and VrefL.
kADC16_ReferenceVoltageSourceValt For alternate reference pair of ValtH and ValtL.

5.5.9 enum adc16_hardware_average_mode_t

Enumerator

kADC16_HardwareAverageCount4 For hardware average with 4 samples.
kADC16_HardwareAverageCount8 For hardware average with 8 samples.
kADC16_HardwareAverageCount16 For hardware average with 16 samples.
kADC16_HardwareAverageCount32 For hardware average with 32 samples.
kADC16_HardwareAverageDisabled Disable the hardware average feature.

5.5.10 enum adc16_hardware_compare_mode_t

Enumerator

kADC16_HardwareCompareMode0 x < value1.
kADC16_HardwareCompareMode1 x > value1.
kADC16_HardwareCompareMode2 if value1 <= value2, then x < value1 || x > value2; else, value1 > x > value2.
kADC16_HardwareCompareMode3 if value1 <= value2, then value1 <= x <= value2; else x >= value1 || x <= value2.

5.6 Function Documentation

5.6.1 void ADC16_Init (ADC_Type * *base,* const adc16_config_t * *config*)

Parameters

base	ADC16 peripheral base address.
config	Pointer to configuration structure. See "adc16_config_t".

5.6.2 void ADC16_Deinit (ADC_Type * *base*)

Parameters

base	ADC16 peripheral base address.

5.6.3 void ADC16_GetDefaultConfig (adc16_config_t * *config*)

This function initializes the converter configuration structure with available settings. The default values are as follows.

```
*   config->referenceVoltageSource     = kADC16_ReferenceVoltageSourceVref
      ;
*   config->clockSource                = kADC16_ClockSourceAsynchronousClock
      ;
*   config->enableAsynchronousClock    = true;
*   config->clockDivider               = kADC16_ClockDivider8;
*   config->resolution                 = kADC16_ResolutionSE12Bit;
*   config->longSampleMode             = kADC16_LongSampleDisabled;
*   config->enableHighSpeed            = false;
*   config->enableLowPower             = false;
*   config->enableContinuousConversion = false;
*
```

Parameters

config	Pointer to the configuration structure.

5.6.4 status_t ADC16_DoAutoCalibration (ADC_Type * *base*)

This auto calibration helps to adjust the plus/minus side gain automatically. Execute the calibration before using the converter. Note that the hardware trigger should be used during the calibration.

Parameters

base	ADC16 peripheral base address.

Returns

Execution status.

Return values

kStatus_Success	Calibration is done successfully.
kStatus_Fail	Calibration has failed.

5.6.5 static void ADC16_SetOffsetValue (ADC_Type * *base,* int16_t *value*) [inline], [static]

This offset value takes effect on the conversion result. If the offset value is not zero, the reading result is subtracted by it. Note, the hardware calibration fills the offset value automatically.

Parameters

base	ADC16 peripheral base address.
value	Setting offset value.

5.6.6 static void ADC16_EnableDMA (ADC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	ADC16 peripheral base address.
enable	Switcher of the DMA feature. "true" means enabled, "false" means not enabled.

5.6.7 static void ADC16_EnableHardwareTrigger (ADC_Type * *base,* bool *enable*) [inline], [static]

Parameters

base	ADC16 peripheral base address.
enable	Switcher of the hardware trigger feature. "true" means enabled, "false" means not enabled.

5.6.8 void ADC16_SetChannelMuxMode (ADC_Type * *base*, adc16_channel_mux_mode_t *mode*)

Some sample pins share the same channel index. The channel mux mode decides which pin is used for an indicated channel.

Parameters

base	ADC16 peripheral base address.
mode	Setting channel mux mode. See "adc16_channel_mux_mode_t".

5.6.9 void ADC16_SetHardwareCompareConfig (ADC_Type * *base*, const adc16_hardware_compare_config_t * *config*)

The hardware compare mode provides a way to process the conversion result automatically by using hardware. Only the result in the compare range is available. To compare the range, see "adc16_hardware-_compare_mode_t" or the appopriate reference manual for more information.

Parameters

base	ADC16 peripheral base address.
config	Pointer to the "adc16_hardware_compare_config_t" structure. Passing "NULL" disables the feature.

5.6.10 void ADC16_SetHardwareAverage (ADC_Type * *base*, adc16_hardware_average_mode_t *mode*)

The hardware average mode provides a way to process the conversion result automatically by using hardware. The multiple conversion results are accumulated and averaged internally making them easier to read.

Parameters

base	ADC16 peripheral base address.
mode	Setting the hardware average mode. See "adc16_hardware_average_mode_t".

5.6.11 uint32_t ADC16_GetStatusFlags (ADC_Type ∗ *base*)

Parameters

base	ADC16 peripheral base address.

Returns

Flags' mask if indicated flags are asserted. See "_adc16_status_flags".

5.6.12 void ADC16_ClearStatusFlags (ADC_Type ∗ *base,* uint32_t *mask*)

Parameters

base	ADC16 peripheral base address.
mask	Mask value for the cleared flags. See "_adc16_status_flags".

5.6.13 void ADC16_SetChannelConfig (ADC_Type ∗ *base,* uint32_t *channelGroup,* const adc16_channel_config_t ∗ *config*)

This operation triggers the conversion when in software trigger mode. When in hardware trigger mode, this API configures the channel while the external trigger source helps to trigger the conversion.

Note that the "Channel Group" has a detailed description. To allow sequential conversions of the ADC to be triggered by internal peripherals, the ADC has more than one group of status and control registers, one for each conversion. The channel group parameter indicates which group of registers are used, for example, channel group 0 is for Group A registers and channel group 1 is for Group B registers. The channel groups are used in a "ping-pong" approach to control the ADC operation. At any point, only one of the channel groups is actively controlling ADC conversions. The channel group 0 is used for both software and hardware trigger modes. Channel group 1 and greater indicates multiple channel group registers for use only in hardware trigger mode. See the chip configuration information in the appropriate MCU reference manual for the number of SC1n registers (channel groups) specific to this device. Channel group 1 or greater are not used for software trigger operation. Therefore, writing to these channel groups does not initiate a new conversion. Updating the channel group 0 while a different channel group is

actively controlling a conversion is allowed and vice versa. Writing any of the channel group registers while that specific channel group is actively controlling a conversion aborts the current conversion.

Parameters

base	ADC16 peripheral base address.
channelGroup	Channel group index.
config	Pointer to the "adc16_channel_config_t" structure for the conversion channel.

5.6.14 static uint32_t ADC16_GetChannelConversionValue (ADC_Type ∗ *base,* uint32_t *channelGroup*) `[inline], [static]`

Parameters

base	ADC16 peripheral base address.
channelGroup	Channel group index.

Returns

Conversion value.

5.6.15 uint32_t ADC16_GetChannelStatusFlags (ADC_Type ∗ *base,* uint32_t *channelGroup*)

Parameters

base	ADC16 peripheral base address.
channelGroup	Channel group index.

Returns

Flags' mask if indicated flags are asserted. See "_adc16_channel_status_flags".

Chapter 6
CMP: Analog Comparator Driver

6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Analog Comparator (CMP) module of MCUXpresso SDK devices.

The CMP driver is a basic comparator with advanced features. The APIs for the basic comparator enable the CMP to compare the two voltages of the two input channels and create the output of the comparator result. The APIs for advanced features can be used as the plug-in functions based on the basic comparator. They can process the comparator's output with hardware support.

6.2 Typical use case

6.2.1 Polling Configuration

```
int main(void)
{
    cmp_config_t mCmpConfigStruct;
    cmp_dac_config_t mCmpDacConfigStruct;

    // ...

    // Configures the comparator.
    CMP_Init(DEMO_CMP_INSTANCE);
    CMP_GetDefaultConfig(&mCmpConfigStruct);
    CMP_Configure(DEMO_CMP_INSTANCE, &mCmpConfigStruct);

    // Configures the DAC channel.
    mCmpDacConfigStruct.referenceVoltageSource =
      kCMP_VrefSourceVin2; // VCC.
    mCmpDacConfigStruct.DACValue = 32U; // Half voltage of logic high-level.
    CMP_SetDACConfig(DEMO_CMP_INSTANCE, &mCmpDacConfigStruct);
    CMP_SetInputChannels(DEMO_CMP_INSTANCE, DEMO_CMP_USER_CHANNEL, DEMO_CMP_DAC_CHANNEL
      );

    while (1)
    {
        if (0U != (kCMP_OutputAssertEventFlag &
      CMP_GetStatusFlags(DEMO_CMP_INSTANCE)))
        {
            // Do something.
        }
        else
        {
            // Do something.
        }
    }
}
```

6.2.2 Interrupt Configuration

```
volatile uint32_t g_CmpFlags = 0U;

// ...

void DEMO_CMP_IRQ_HANDLER_FUNC(void)
{
    g_CmpFlags = CMP_GetStatusFlags(DEMO_CMP_INSTANCE);
    CMP_ClearStatusFlags(DEMO_CMP_INSTANCE, kCMP_OutputRisingEventFlag |
      kCMP_OutputFallingEventFlag);
    if (0U != (g_CmpFlags & kCMP_OutputRisingEventFlag))
    {
        // Do something.
    }
    else if (0U != (g_CmpFlags & kCMP_OutputFallingEventFlag))
    {
        // Do something.
    }
}

int main(void)
{
    cmp_config_t mCmpConfigStruct;
    cmp_dac_config_t mCmpDacConfigStruct;

    // ...
    EnableIRQ(DEMO_CMP_IRQ_ID);
    // ...

    // Configures the comparator.
    CMP_Init(DEMO_CMP_INSTANCE);
    CMP_GetDefaultConfig(&mCmpConfigStruct);
    CMP_Configure(DEMO_CMP_INSTANCE, &mCmpConfigStruct);

    // Configures the DAC channel.
    mCmpDacConfigStruct.referenceVoltageSource =
      kCMP_VrefSourceVin2; // VCC.
    mCmpDacConfigStruct.DACValue = 32U; // Half voltage of logic high-level.
    CMP_SetDACConfig(DEMO_CMP_INSTANCE, &mCmpDacConfigStruct);
    CMP_SetInputChannels(DEMO_CMP_INSTANCE, DEMO_CMP_USER_CHANNEL, DEMO_CMP_DAC_CHANNEL
      );

    // Enables the output rising and falling interrupts.
    CMP_EnableInterrupts(DEMO_CMP_INSTANCE,
      kCMP_OutputRisingInterruptEnable |
      kCMP_OutputFallingInterruptEnable);

    while (1)
    {
    }
}
```

Data Structures

- struct cmp_config_t
  Configures the comparator. More...
- struct cmp_filter_config_t
  Configures the filter. More...
- struct cmp_dac_config_t
  Configures the internal DAC. More...

Enumerations

- enum _cmp_interrupt_enable {
  kCMP_OutputRisingInterruptEnable = CMP_SCR_IER_MASK,
  kCMP_OutputFallingInterruptEnable = CMP_SCR_IEF_MASK }
  Interrupt enable/disable mask.
- enum _cmp_status_flags {
  kCMP_OutputRisingEventFlag = CMP_SCR_CFR_MASK,
  kCMP_OutputFallingEventFlag = CMP_SCR_CFF_MASK,
  kCMP_OutputAssertEventFlag = CMP_SCR_COUT_MASK }
  Status flags' mask.
- enum cmp_hysteresis_mode_t {
  kCMP_HysteresisLevel0 = 0U,
  kCMP_HysteresisLevel1 = 1U,
  kCMP_HysteresisLevel2 = 2U,
  kCMP_HysteresisLevel3 = 3U }
  CMP Hysteresis mode.
- enum cmp_reference_voltage_source_t {
  kCMP_VrefSourceVin1 = 0U,
  kCMP_VrefSourceVin2 = 1U }
  CMP Voltage Reference source.

Driver version

- #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  CMP driver version 2.0.0.

Initialization

- void CMP_Init (CMP_Type ∗base, const cmp_config_t ∗config)
  Initializes the CMP.
- void CMP_Deinit (CMP_Type ∗base)
  De-initializes the CMP module.
- static void CMP_Enable (CMP_Type ∗base, bool enable)
  Enables/disables the CMP module.
- void CMP_GetDefaultConfig (cmp_config_t ∗config)
  Initializes the CMP user configuration structure.
- void CMP_SetInputChannels (CMP_Type ∗base, uint8_t positiveChannel, uint8_t negativeChannel)
  Sets the input channels for the comparator.

Advanced Features

- void CMP_EnableDMA (CMP_Type ∗base, bool enable)
  Enables/disables the DMA request for rising/falling events.
- void CMP_SetFilterConfig (CMP_Type ∗base, const cmp_filter_config_t ∗config)
  Configures the filter.
- void CMP_SetDACConfig (CMP_Type ∗base, const cmp_dac_config_t ∗config)
  Configures the internal DAC.
- void CMP_EnableInterrupts (CMP_Type ∗base, uint32_t mask)
  Enables the interrupts.

- void CMP_DisableInterrupts (CMP_Type ∗base, uint32_t mask)
  Disables the interrupts.

Results

- uint32_t CMP_GetStatusFlags (CMP_Type ∗base)
  Gets the status flags.
- void CMP_ClearStatusFlags (CMP_Type ∗base, uint32_t mask)
  Clears the status flags.

6.3 Data Structure Documentation

6.3.1 struct cmp_config_t

Data Fields

- bool enableCmp
  Enable the CMP module.
- cmp_hysteresis_mode_t hysteresisMode
  CMP Hysteresis mode.
- bool enableHighSpeed
  Enable High-speed (HS) comparison mode.
- bool enableInvertOutput
  Enable the inverted comparator output.
- bool useUnfilteredOutput
  Set the compare output(COUT) to equal COUTA(true) or COUT(false).
- bool enablePinOut
  The comparator output is available on the associated pin.
- bool enableTriggerMode
  Enable the trigger mode.

6.3.1.0.0.4 Field Documentation

6.3.1.0.0.4.1 bool cmp_config_t::enableCmp

6.3.1.0.0.4.2 cmp_hysteresis_mode_t cmp_config_t::hysteresisMode

6.3.1.0.0.4.3 bool cmp_config_t::enableHighSpeed

6.3.1.0.0.4.4 bool cmp_config_t::enableInvertOutput

6.3.1.0.0.4.5 bool cmp_config_t::useUnfilteredOutput

6.3.1.0.0.4.6 bool cmp_config_t::enablePinOut

6.3.1.0.0.4.7 bool cmp_config_t::enableTriggerMode

6.3.2 struct cmp_filter_config_t

Data Fields

- uint8_t filterCount
  Filter Sample Count.
- uint8_t filterPeriod
  Filter Sample Period.

6.3.2.0.0.5 Field Documentation

6.3.2.0.0.5.1 uint8_t cmp_filter_config_t::filterCount

Available range is 1-7; 0 disables the filter.

6.3.2.0.0.5.2 uint8_t cmp_filter_config_t::filterPeriod

The divider to the bus clock. Available range is 0-255.

6.3.3 struct cmp_dac_config_t

Data Fields

- cmp_reference_voltage_source_t referenceVoltageSource
  Supply voltage reference source.
- uint8_t DACValue
  Value for the DAC Output Voltage.

6.3.3.0.0.6 Field Documentation

6.3.3.0.0.6.1 cmp_reference_voltage_source_t cmp_dac_config_t::referenceVoltageSource

6.3.3.0.0.6.2 uint8_t cmp_dac_config_t::DACValue

Available range is 0-63.

6.4 Macro Definition Documentation

6.4.1 #define FSL_CMP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

6.5 Enumeration Type Documentation

6.5.1 enum _cmp_interrupt_enable

Enumerator

kCMP_OutputRisingInterruptEnable Comparator interrupt enable rising.
kCMP_OutputFallingInterruptEnable Comparator interrupt enable falling.

6.5.2 enum _cmp_status_flags

Enumerator

kCMP_OutputRisingEventFlag Rising-edge on the comparison output has occurred.
kCMP_OutputFallingEventFlag Falling-edge on the comparison output has occurred.
kCMP_OutputAssertEventFlag Return the current value of the analog comparator output.

6.5.3 enum cmp_hysteresis_mode_t

Enumerator

kCMP_HysteresisLevel0 Hysteresis level 0.
kCMP_HysteresisLevel1 Hysteresis level 1.
kCMP_HysteresisLevel2 Hysteresis level 2.
kCMP_HysteresisLevel3 Hysteresis level 3.

6.5.4 enum cmp_reference_voltage_source_t

Enumerator

kCMP_VrefSourceVin1 Vin1 is selected as a resistor ladder network supply reference Vin.
kCMP_VrefSourceVin2 Vin2 is selected as a resistor ladder network supply reference Vin.

6.6 Function Documentation

6.6.1 void CMP_Init (CMP_Type * *base,* const cmp_config_t * *config*)

This function initializes the CMP module. The operations included are as follows.

- Enabling the clock for CMP module.
- Configuring the comparator.
- Enabling the CMP module. Note that for some devices, multiple CMP instances share the same clock gate. In this case, to enable the clock for any instance enables all CMPs. See the appropriate MCU reference manual for the clock assignment of the CMP.

Parameters

base	CMP peripheral base address.
config	Pointer to the configuration structure.

6.6.2 void CMP_Deinit (CMP_Type * *base*)

This function de-initializes the CMP module. The operations included are as follows.

- Disabling the CMP module.
- Disabling the clock for CMP module.

This function disables the clock for the CMP. Note that for some devices, multiple CMP instances share the same clock gate. In this case, before disabling the clock for the CMP, ensure that all the CMP instances are not used.

Parameters

base	CMP peripheral base address.

6.6.3 static void CMP_Enable (CMP_Type * *base,* bool *enable*) `[inline], [static]`

Parameters

base	CMP peripheral base address.

enable	Enables or disables the module.

6.6.4 void CMP_GetDefaultConfig (cmp_config_t * *config*)

This function initializes the user configuration structure to these default values.

```
*   config->enableCmp           = true;
*   config->hysteresisMode      = kCMP_HysteresisLevel0;
*   config->enableHighSpeed     = false;
*   config->enableInvertOutput  = false;
*   config->useUnfilteredOutput = false;
*   config->enablePinOut        = false;
*   config->enableTriggerMode   = false;
*
```

Parameters

config	Pointer to the configuration structure.

6.6.5 void CMP_SetInputChannels (CMP_Type * *base,* uint8_t *positiveChannel,* uint8_t *negativeChannel*)

This function sets the input channels for the comparator. Note that two input channels cannot be set the same way in the application. When the user selects the same input from the analog mux to the positive and negative port, the comparator is disabled automatically.

Parameters

base	CMP peripheral base address.
positive-Channel	Positive side input channel number. Available range is 0-7.
negative-Channel	Negative side input channel number. Available range is 0-7.

6.6.6 void CMP_EnableDMA (CMP_Type * *base,* bool *enable*)

This function enables/disables the DMA request for rising/falling events. Either event triggers the generation of the DMA request from CMP if the DMA feature is enabled. Both events are ignored for generating the DMA request from the CMP if the DMA is disabled.

Parameters

base	CMP peripheral base address.
enable	Enables or disables the feature.

6.6.7 void CMP_SetFilterConfig (CMP_Type * *base,* const cmp_filter_config_t * *config*)

Parameters

base	CMP peripheral base address.
config	Pointer to the configuration structure.

6.6.8 void CMP_SetDACConfig (CMP_Type * *base,* const cmp_dac_config_t * *config*)

Parameters

base	CMP peripheral base address.
config	Pointer to the configuration structure. "NULL" disables the feature.

6.6.9 void CMP_EnableInterrupts (CMP_Type * *base,* uint32_t *mask*)

Parameters

base	CMP peripheral base address.
mask	Mask value for interrupts. See "_cmp_interrupt_enable".

6.6.10 void CMP_DisableInterrupts (CMP_Type * *base,* uint32_t *mask*)

Parameters

base	CMP peripheral base address.
mask	Mask value for interrupts. See "_cmp_interrupt_enable".

6.6.11 uint32_t CMP_GetStatusFlags (CMP_Type ∗ *base*)

Parameters

base	CMP peripheral base address.

Returns

Mask value for the asserted flags. See "_cmp_status_flags".

6.6.12 void CMP_ClearStatusFlags (CMP_Type ∗ *base,* uint32_t *mask*)

Parameters

base	CMP peripheral base address.
mask	Mask value for the flags. See "_cmp_status_flags".

Chapter 7
CMT: Carrier Modulator Transmitter Driver

7.1 Overview

The carrier modulator transmitter (CMT) module provides the means to generate the protocol timing and carrier signals for a side variety of encoding schemes. The CMT incorporates hardware to off-load the critical and/or lengthy timing requirements associated with signal generation from the CPU. The MCUXpresso SDK provides a driver for the CMT module of the MCUXpresso SDK devices.

7.2 Clock formulas

The CMT module has internal clock dividers. It was originally designed to be based on an 8 MHz bus clock that can be divided by 1, 2, 4, or 8 according to the specification. To be compatible with a higher bus frequency, the primary prescaler (PPS) was developed to receive a higher frequency and generate a clock enable signal called an intermediate frequency (IF). The IF must be approximately equal to 8 MHz and works as a clock enable to the secondary prescaler. For the PPS, the prescaler is selected according to the bus clock to generate an intermediate clock approximate to 8 MHz and is selected as (bus_clock_hz/8000000). The secondary prescaler is the "cmtDivider". The clocks for the CMT module are listed below.

1. CMT clock frequency = bus_clock_Hz / (bus_clock_Hz / 8000000) / cmtDivider
2. CMT carrier and generator frequency = CMT clock frequency / (highCount1 + lowCount1)
   (In FSK mode, the second frequency = CMT clock frequency / (highCount2 + lowCount2))
3. CMT infrared output signal frequency
   a. In Time and Baseband mode
   CMT IRO signal mark time = (markCount + 1) / (CMT clock frequency / 8)
   CMT IRO signal space time = spaceCount / (CMT clock frequency / 8)
   b. In FSK mode
   CMT IRO signal mark time = (markCount + 1) / CMT carrier and generator frequency
   CMT IRO signal space time = spaceCount / CMT carrier and generator frequency

7.3 Typical use case

This is an example code to initialize data.

```
cmt_config_t config;
cmt_modulate_config_t modulateConfig;
uint32_t busClock;

// Gets the bus clock for the CMT module.
busClock = CLOCK_GetFreq(kCLOCK_BusClk);

CMT_GetDefaultConfig(&config);

// Interrupts is enabled to change the modulate mark and space count.
config.isInterruptEnabled = true;
```

Typical use case

```
CMT_Init(CMT, &config, busClock);

// Prepares the modulate configuration for a use case.
modulateConfig.highCount1 = ...;
modulateConfig.lowCount1 = ...;
modulateConfig.markCount = ...;
modulateConfig.spaceCount = ...;

// Sets the time mode.
CMT_SetMode(CMT, kCMT_TimeMode, &modulateConfig);
```

This is an example IRQ handler to change the mark and space count to complete data modulation.

```
// The data length has been transmitted.
uint32_t g_CmtDataBitLen;

void CMT_IRQHandler(void)
{
    if (CMT_GetStatusFlags(CMT))
    {
        if (g_CmtDataBitLen <= CMT_TEST_DATA_BITS)
        {
            // LSB.
            if (data & ((uint32_t)0x01 << g_CmtDataBitLen))
            {
                CMT_SetModulateMarkSpace(CMT, g_CmtModDataOneMarkCount,
      g_CmtModDataOneSpaceCount);
            }
            else
            {
                CMT_SetModulateMarkSpace(CMT, g_CmtModDataZeroMarkCount,
      g_CmtModDataZeroSpaceCount);
            }
        }
    }
}
```

Data Structures

- struct cmt_modulate_config_t
  CMT carrier generator and modulator configuration structure. More...
- struct cmt_config_t
  CMT basic configuration structure. More...

Enumerations

- enum cmt_mode_t {
  kCMT_DirectIROCtl = 0x00U,
  kCMT_TimeMode = 0x01U,
  kCMT_FSKMode = 0x05U,
  kCMT_BasebandMode = 0x09U }
  The modes of CMT.
- enum cmt_primary_clkdiv_t {

kCMT_PrimaryClkDiv1 = 0U,
kCMT_PrimaryClkDiv2 = 1U,
kCMT_PrimaryClkDiv3 = 2U,
kCMT_PrimaryClkDiv4 = 3U,
kCMT_PrimaryClkDiv5 = 4U,
kCMT_PrimaryClkDiv6 = 5U,
kCMT_PrimaryClkDiv7 = 6U,
kCMT_PrimaryClkDiv8 = 7U,
kCMT_PrimaryClkDiv9 = 8U,
kCMT_PrimaryClkDiv10 = 9U,
kCMT_PrimaryClkDiv11 = 10U,
kCMT_PrimaryClkDiv12 = 11U,
kCMT_PrimaryClkDiv13 = 12U,
kCMT_PrimaryClkDiv14 = 13U,
kCMT_PrimaryClkDiv15 = 14U,
kCMT_PrimaryClkDiv16 = 15U }

The CMT clock divide primary prescaler.

- enum cmt_second_clkdiv_t {
  kCMT_SecondClkDiv1 = 0U,
  kCMT_SecondClkDiv2 = 1U,
  kCMT_SecondClkDiv4 = 2U,
  kCMT_SecondClkDiv8 = 3U }

  The CMT clock divide secondary prescaler.
- enum cmt_infrared_output_polarity_t {
  kCMT_IROActiveLow = 0U,
  kCMT_IROActiveHigh = 1U }

  The CMT infrared output polarity.
- enum cmt_infrared_output_state_t {
  kCMT_IROCtlLow = 0U,
  kCMT_IROCtlHigh = 1U }

  The CMT infrared output signal state control.
- enum _cmt_interrupt_enable { kCMT_EndOfCycleInterruptEnable = CMT_MSC_EOCIE_MASK }

  CMT interrupt configuration structure, default settings all disabled.

Driver version

- #define FSL_CMT_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

  CMT driver version 2.0.1.

Initialization and deinitialization

- void CMT_GetDefaultConfig (cmt_config_t *config)

  Gets the CMT default configuration structure.
- void CMT_Init (CMT_Type *base, const cmt_config_t *config, uint32_t busClock_Hz)

  Initializes the CMT module.
- void CMT_Deinit (CMT_Type *base)

Disables the CMT module and gate control.

Basic Control Operations

- void CMT_SetMode (CMT_Type ∗base, cmt_mode_t mode, cmt_modulate_config_t ∗modulateConfig)
  Selects the mode for CMT.
- cmt_mode_t CMT_GetMode (CMT_Type ∗base)
  Gets the mode of the CMT module.
- uint32_t CMT_GetCMTFrequency (CMT_Type ∗base, uint32_t busClock_Hz)
  Gets the actual CMT clock frequency.
- static void CMT_SetCarrirGenerateCountOne (CMT_Type ∗base, uint32_t highCount, uint32_t lowCount)
  Sets the primary data set for the CMT carrier generator counter.
- static void CMT_SetCarrirGenerateCountTwo (CMT_Type ∗base, uint32_t highCount, uint32_t lowCount)
  Sets the secondary data set for the CMT carrier generator counter.
- void CMT_SetModulateMarkSpace (CMT_Type ∗base, uint32_t markCount, uint32_t spaceCount)
  Sets the modulation mark and space time period for the CMT modulator.
- static void CMT_EnableExtendedSpace (CMT_Type ∗base, bool enable)
  Enables or disables the extended space operation.
- void CMT_SetIroState (CMT_Type ∗base, cmt_infrared_output_state_t state)
  Sets the IRO (infrared output) signal state.
- static void CMT_EnableInterrupts (CMT_Type ∗base, uint32_t mask)
  Enables the CMT interrupt.
- static void CMT_DisableInterrupts (CMT_Type ∗base, uint32_t mask)
  Disables the CMT interrupt.
- static uint32_t CMT_GetStatusFlags (CMT_Type ∗base)
  Gets the end of the cycle status flag.

7.4 Data Structure Documentation

7.4.1 struct cmt_modulate_config_t

Data Fields

- uint8_t highCount1
  The high-time for carrier generator first register.
- uint8_t lowCount1
  The low-time for carrier generator first register.
- uint8_t highCount2
  The high-time for carrier generator second register for FSK mode.
- uint8_t lowCount2
  The low-time for carrier generator second register for FSK mode.
- uint16_t markCount
  The mark time for the modulator gate.
- uint16_t spaceCount
  The space time for the modulator gate.

7.4.1.0.0.7 Field Documentation

7.4.1.0.0.7.1 uint8_t cmt_modulate_config_t::highCount1

7.4.1.0.0.7.2 uint8_t cmt_modulate_config_t::lowCount1

7.4.1.0.0.7.3 uint8_t cmt_modulate_config_t::highCount2

7.4.1.0.0.7.4 uint8_t cmt_modulate_config_t::lowCount2

7.4.1.0.0.7.5 uint16_t cmt_modulate_config_t::markCount

7.4.1.0.0.7.6 uint16_t cmt_modulate_config_t::spaceCount

7.4.2 struct cmt_config_t

Data Fields

- bool isInterruptEnabled
  Timer interrupt 0-disable, 1-enable.
- bool isIroEnabled
  The IRO output 0-disabled, 1-enabled.
- cmt_infrared_output_polarity_t iroPolarity
  The IRO polarity.
- cmt_second_clkdiv_t divider
  The CMT clock divide prescaler.

7.4.2.0.0.8 Field Documentation

7.4.2.0.0.8.1 bool cmt_config_t::isInterruptEnabled

7.4.2.0.0.8.2 bool cmt_config_t::isIroEnabled

7.4.2.0.0.8.3 cmt_infrared_output_polarity_t cmt_config_t::iroPolarity

7.4.2.0.0.8.4 cmt_second_clkdiv_t cmt_config_t::divider

7.5 Macro Definition Documentation

7.5.1 #define FSL_CMT_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

7.6 Enumeration Type Documentation

7.6.1 enum cmt_mode_t

Enumerator

kCMT_DirectIROCtl Carrier modulator is disabled and the IRO signal is directly in software control.

kCMT_TimeMode Carrier modulator is enabled in time mode.

kCMT_FSKMode Carrier modulator is enabled in FSK mode.
kCMT_BasebandMode Carrier modulator is enabled in baseband mode.

7.6.2 enum cmt_primary_clkdiv_t

The primary clock divider is used to divider the bus clock to get the intermediate frequency to approximately equal to 8 MHZ. When the bus clock is 8 MHZ, set primary prescaler to "kCMT_PrimaryClkDiv1".

Enumerator

kCMT_PrimaryClkDiv1 The intermediate frequency is the bus clock divided by 1.
kCMT_PrimaryClkDiv2 The intermediate frequency is the bus clock divided by 2.
kCMT_PrimaryClkDiv3 The intermediate frequency is the bus clock divided by 3.
kCMT_PrimaryClkDiv4 The intermediate frequency is the bus clock divided by 4.
kCMT_PrimaryClkDiv5 The intermediate frequency is the bus clock divided by 5.
kCMT_PrimaryClkDiv6 The intermediate frequency is the bus clock divided by 6.
kCMT_PrimaryClkDiv7 The intermediate frequency is the bus clock divided by 7.
kCMT_PrimaryClkDiv8 The intermediate frequency is the bus clock divided by 8.
kCMT_PrimaryClkDiv9 The intermediate frequency is the bus clock divided by 9.
kCMT_PrimaryClkDiv10 The intermediate frequency is the bus clock divided by 10.
kCMT_PrimaryClkDiv11 The intermediate frequency is the bus clock divided by 11.
kCMT_PrimaryClkDiv12 The intermediate frequency is the bus clock divided by 12.
kCMT_PrimaryClkDiv13 The intermediate frequency is the bus clock divided by 13.
kCMT_PrimaryClkDiv14 The intermediate frequency is the bus clock divided by 14.
kCMT_PrimaryClkDiv15 The intermediate frequency is the bus clock divided by 15.
kCMT_PrimaryClkDiv16 The intermediate frequency is the bus clock divided by 16.

7.6.3 enum cmt_second_clkdiv_t

The second prescaler can be used to divide the 8 MHZ CMT clock by 1, 2, 4, or 8 according to the specification.

Enumerator

kCMT_SecondClkDiv1 The CMT clock is the intermediate frequency frequency divided by 1.
kCMT_SecondClkDiv2 The CMT clock is the intermediate frequency frequency divided by 2.
kCMT_SecondClkDiv4 The CMT clock is the intermediate frequency frequency divided by 4.
kCMT_SecondClkDiv8 The CMT clock is the intermediate frequency frequency divided by 8.

7.6.4 enum cmt_infrared_output_polarity_t

Enumerator

kCMT_IROActiveLow The CMT infrared output signal polarity is active-low.
kCMT_IROActiveHigh The CMT infrared output signal polarity is active-high.

7.6.5 enum cmt_infrared_output_state_t

Enumerator

kCMT_IROCtlLow The CMT Infrared output signal state is controlled to low.
kCMT_IROCtlHigh The CMT Infrared output signal state is controlled to high.

7.6.6 enum _cmt_interrupt_enable

This structure contains the settings for all of the CMT interrupt configurations.

Enumerator

kCMT_EndOfCycleInterruptEnable CMT end of cycle interrupt.

7.7 Function Documentation

7.7.1 void CMT_GetDefaultConfig (cmt_config_t ∗ *config*)

This API gets the default configuration structure for the CMT_Init(). Use the initialized structure unchanged in CMT_Init() or modify fields of the structure before calling the CMT_Init().

Parameters

config	The CMT configuration structure pointer.

7.7.2 void CMT_Init (CMT_Type ∗ *base,* const cmt_config_t ∗ *config,* uint32_t *busClock_Hz*)

This function ungates the module clock and sets the CMT internal clock, interrupt, and infrared output signal for the CMT module.

Parameters

base	CMT peripheral base address.
config	The CMT basic configuration structure.
busClock_Hz	The CMT module input clock - bus clock frequency.

7.7.3 void CMT_Deinit (CMT_Type * *base*)

This function disables CMT modulator, interrupts, and gates the CMT clock control. CMT_Init must be called to use the CMT again.

Parameters

base	CMT peripheral base address.

7.7.4 void CMT_SetMode (CMT_Type * *base*, cmt_mode_t *mode*, cmt_modulate_config_t * *modulateConfig*)

Parameters

base	CMT peripheral base address.
mode	The CMT feature mode enumeration. See "cmt_mode_t".
modulate-Config	The carrier generation and modulator configuration.

7.7.5 cmt_mode_t CMT_GetMode (CMT_Type * *base*)

Parameters

base	CMT peripheral base address.

Returns

The CMT mode. kCMT_DirectIROCtl Carrier modulator is disabled; the IRO signal is directly in software control. kCMT_TimeMode Carrier modulator is enabled in time mode. kCMT_FSKMode Carrier modulator is enabled in FSK mode. kCMT_BasebandMode Carrier modulator is enabled in baseband mode.

7.7.6 uint32_t CMT_GetCMTFrequency (CMT_Type * *base,* uint32_t *busClock_Hz*)

Parameters

base	CMT peripheral base address.
busClock_Hz	CMT module input clock - bus clock frequency.

Returns

The CMT clock frequency.

7.7.7 static void CMT_SetCarrirGenerateCountOne (CMT_Type ∗ *base,* uint32_t *highCount,* uint32_t *lowCount*) [inline], [static]

This function sets the high-time and low-time of the primary data set for the CMT carrier generator counter to control the period and the duty cycle of the output carrier signal. If the CMT clock period is Tcmt, the period of the carrier generator signal equals (highCount + lowCount) ∗ Tcmt. The duty cycle equals to highCount / (highCount + lowCount).

Parameters

base	CMT peripheral base address.
highCount	The number of CMT clocks for carrier generator signal high time, integer in the range of 1 ∼ 0xFF.
lowCount	The number of CMT clocks for carrier generator signal low time, integer in the range of 1 ∼ 0xFF.

7.7.8 static void CMT_SetCarrirGenerateCountTwo (CMT_Type ∗ *base,* uint32_t *highCount,* uint32_t *lowCount*) [inline], [static]

This function is used for FSK mode setting the high-time and low-time of the secondary data set CMT carrier generator counter to control the period and the duty cycle of the output carrier signal. If the CMT clock period is Tcmt, the period of the carrier generator signal equals (highCount + lowCount) ∗ Tcmt. The duty cycle equals highCount / (highCount + lowCount).

Parameters

base	CMT peripheral base address.

highCount	The number of CMT clocks for carrier generator signal high time, integer in the range of 1 ∼ 0xFF.
lowCount	The number of CMT clocks for carrier generator signal low time, integer in the range of 1 ∼ 0xFF.

7.7.9 void CMT_SetModulateMarkSpace (CMT_Type ∗ *base,* uint32_t *markCount,* uint32_t *spaceCount*)

This function sets the mark time period of the CMT modulator counter to control the mark time of the output modulated signal from the carrier generator output signal. If the CMT clock frequency is Fcmt and the carrier out signal frequency is fcg:

- In Time and Baseband mode: The mark period of the generated signal equals (markCount + 1) / (Fcmt/8). The space period of the generated signal equals spaceCount / (Fcmt/8).
- In FSK mode: The mark period of the generated signal equals (markCount + 1)/fcg. The space period of the generated signal equals spaceCount / fcg.

Parameters

base	Base address for current CMT instance.
markCount	The number of clock period for CMT modulator signal mark period, in the range of 0 ∼ 0xFFFF.
spaceCount	The number of clock period for CMT modulator signal space period, in the range of the 0 ∼ 0xFFFF.

7.7.10 static void CMT_EnableExtendedSpace (CMT_Type ∗ *base,* bool *enable*) `[inline],[static]`

This function is used to make the space period longer for time, baseband, and FSK modes.

Parameters

base	CMT peripheral base address.
enable	True enable the extended space, false disable the extended space.

7.7.11 void CMT_SetIroState (CMT_Type * *base,* cmt_infrared_output_state_t *state*)

Changes the states of the IRO signal when the kCMT_DirectIROMode mode is set and the IRO signal is enabled.

Parameters

base	CMT peripheral base address.
state	The control of the IRO signal. See "cmt_infrared_output_state_t"

7.7.12 static void CMT_EnableInterrupts (CMT_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the CMT interrupts according to the provided mask if enabled. The CMT only has the end of the cycle interrupt - an interrupt occurs at the end of the modulator cycle. This interrupt provides a means for the user to reload the new mark/space values into the CMT modulator data registers and verify the modulator mark and space. For example, to enable the end of cycle, do the following.

```
*     CMT_EnableInterrupts(CMT,
      kCMT_EndOfCycleInterruptEnable);
*
```

Parameters

base	CMT peripheral base address.
mask	The interrupts to enable. Logical OR of _cmt_interrupt_enable.

7.7.13 static void CMT_DisableInterrupts (CMT_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the CMT interrupts according to the provided maskIf enabled. The CMT only has the end of the cycle interrupt. For example, to disable the end of cycle, do the following.

```
*     CMT_DisableInterrupts(CMT,
      kCMT_EndOfCycleInterruptEnable);
*
```

Parameters

base	CMT peripheral base address.
mask	The interrupts to enable. Logical OR of _cmt_interrupt_enable.

7.7.14 static uint32_t CMT_GetStatusFlags (CMT_Type * *base*) [inline], [static]

The flag is set:

- When the modulator is not currently active and carrier and modulator are set to start the initial CMT transmission.
- At the end of each modulation cycle when the counter is reloaded and the carrier and modulator are enabled.

Parameters

base	CMT peripheral base address.

Returns

Current status of the end of cycle status flag
- non-zero: End-of-cycle has occurred.
- zero: End-of-cycle has not yet occurred since the flag last cleared.

Chapter 8
COP: Watchdog Driver

8.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Computer Operating Properly module (COP) of MCUXpresso SDK devices.

8.2 Typical use case

```
cop_config_t config;
COP_GetDefaultConfig(&config);
config.timeoutCycles = kCOP_2Power8CyclesOr2Power16Cycles;
COP_Init(sim_base,&config);
```

Data Structures

- struct cop_config_t
  Describes COP configuration structure. More...

Enumerations

- enum cop_clock_source_t {
  kCOP_LpoClock = 0U,
  kCOP_McgIrClock = 1U,
  kCOP_OscErClock = 2U,
  kCOP_BusClock = 3U }
  COP clock source selection.
- enum cop_timeout_cycles_t {
  kCOP_2Power5CyclesOr2Power13Cycles = 1U,
  kCOP_2Power8CyclesOr2Power16Cycles = 2U,
  kCOP_2Power10CyclesOr2Power18Cycles = 3U }
  Define the COP timeout cycles.
- enum cop_timeout_mode_t {
  kCOP_ShortTimeoutMode = 0U,
  kCOP_LongTimeoutMode = 1U }
  Define the COP timeout mode.

Driver version

- #define FSL_COP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  COP driver version 2.0.0.

COP refresh sequence.

- #define COP_FIRST_BYTE_OF_REFRESH (0x55U)

First byte of refresh sequence.
- #define COP_SECOND_BYTE_OF_REFRESH (0xAAU)
  Second byte of refresh sequence.

COP Functional Operation

- void COP_GetDefaultConfig (cop_config_t *config)
  Initializes the COP configuration structure.
- void COP_Init (SIM_Type *base, const cop_config_t *config)
  Initializes the COP module.
- static void COP_Disable (SIM_Type *base)
  De-initializes the COP module.
- void COP_Refresh (SIM_Type *base)
  Refreshes the COP timer.

8.3 Data Structure Documentation

8.3.1 struct cop_config_t

Data Fields

- bool enableWindowMode
  COP run mode: window mode or normal mode.
- cop_timeout_mode_t timeoutMode
  COP timeout mode: long timeout or short timeout.
- bool enableStop
  Enable or disable COP in STOP mode.
- bool enableDebug
  Enable or disable COP in DEBUG mode.
- cop_clock_source_t clockSource
  Set COP clock source.
- cop_timeout_cycles_t timeoutCycles
  Set COP timeout value.

8.4 Macro Definition Documentation

8.4.1 #define FSL_COP_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

8.5 Enumeration Type Documentation

8.5.1 enum cop_clock_source_t

Enumerator

kCOP_LpoClock COP clock sourced from LPO.
kCOP_McgIrClock COP clock sourced from MCGIRCLK.
kCOP_OscErClock COP clock sourced from OSCERCLK.
kCOP_BusClock COP clock sourced from Bus clock.

8.5.2 enum cop_timeout_cycles_t

Enumerator

kCOP_2Power5CyclesOr2Power13Cycles 2^5 or 2^13 clock cycles
kCOP_2Power8CyclesOr2Power16Cycles 2^8 or 2^16 clock cycles
kCOP_2Power10CyclesOr2Power18Cycles 2^10 or 2^18 clock cycles

8.5.3 enum cop_timeout_mode_t

Enumerator

kCOP_ShortTimeoutMode COP selects long timeout.
kCOP_LongTimeoutMode COP selects short timeout.

8.6 Function Documentation

8.6.1 void COP_GetDefaultConfig (cop_config_t ∗ *config*)

This function initializes the COP configuration structure to default values. The default values are:

```
*   copConfig->enableWindowMode = false;
*   copConfig->timeoutMode = kCOP_LongTimeoutMode;
*   copConfig->enableStop = false;
*   copConfig->enableDebug = false;
*   copConfig->clockSource = kCOP_LpoClock;
*   copConfig->timeoutCycles = kCOP_2Power10CyclesOr2Power18Cycles;
*
```

Parameters

config	Pointer to the COP configuration structure.

See Also

cop_config_t

8.6.2 void COP_Init (SIM_Type ∗ *base,* const cop_config_t ∗ *config*)

This function configures the COP. After it is called, the COP starts running according to the configuration. Because all COP control registers are write-once only, the COP_Init function and the COP_Disable function can be called only once. A second call has no effect.

Example:

```
*  cop_config_t config;
*  COP_GetDefaultConfig(&config);
*  config.timeoutCycles = kCOP_2Power8CyclesOr2Power16Cycles
      ;
*  COP_Init(sim_base,&config);
*
```

Parameters

base	SIM peripheral base address.
config	The configuration of COP.

8.6.3 static void COP_Disable (SIM_Type * *base*) [inline], [static]

This dedicated function is not provided. Instead, the COP_Disable function can be used to disable the COP.

Disables the COP module.

This function disables the COP Watchdog. Note: The COP configuration register is a write-once after reset. To disable the COP Watchdog, call this function first.

Parameters

base	SIM peripheral base address.

8.6.4 void COP_Refresh (SIM_Type * *base*)

This function feeds the COP.

Parameters

base	SIM peripheral base address.

Chapter 9
DCDC: DCDC Converter

9.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the DCDC Converter (DCDC) module of MCUXpresso SDK devices.

This part describes the programming interface of the DCDC Converter Peripheral driver. The DCDC converter module is a switching mode DC-DC converter supporting Buck, Boost and Bypass mode. It could produce multiple switching outputs for SoC peripherals and external devices with high conversion efficiency. The converter can be operated in continuous or pulsed mode.

As a module to provide the power for hardware system, the DCDC would start working when the system is powered up before the software takes over SoC. Some important configuration like selecting BUCK/BOOST/BYPASS mode is done by board settings. Before software can access the DCDC's registers, DCDC are already working normally with default settings.

However, if the application needs to improve the DCDC's performance or change the default settings, DCDC driver would help. The DCDC's register can not be accessed by software before its initialization (open the clock gate). Then user can configure the hardware according to the application guide from RM.

9.2 Function groups

9.2.1 Initialization and deinitialization

This function group is to enable/disable the operations to DCDC module through the driver.

9.2.2 Status

Provides functions to get and clear the DCDC status.

9.2.3 Interrupts

Provides functions to enable/disable DCDC interrupts.

9.2.4 Misc control

Provides functions to set the DCDC's miscellaneous control.

9.3 Application guideline

9.3.1 Continuous mode

As guideline from RM, to have better efficiency and ripple. The following call is recommended:

```
dcdc_loop_control_config_t dcdcLoopCtrlConfigStruct;

DCDC_GetDefaultLoopControlConfig(&dcdcLoopCtrlConfigStruct);
dcdcLoopCtrlConfigStruct.enableDiffHysteresis = true; //DCDC_LOOPCTRL_EN_DF_HYST = 1
dcdcLoopCtrlConfigStruct.enableCommonHysteresis = true; //DCDC_LOOPCTRL_EN_CM_HYST =
       1
dcdcLoopCtrlConfigStruct.enableInvertHysteresisSign = true; //
      DCDC_LOOPCTRL_HYST_SIGN = 1
DCDC_SetLoopControlConfig(DCDC, &dcdcLoopCtrlConfigStruct);
```

In boost mode, POSLIMIT_BOOST_IN is set to small value by default. To limit startup voltage, set it to 0x12 after startup, to provide high current to output, especially when battery voltage is low. The following call could be used.

```
DCDC_SetUpperLimitDutyCycleBoost(DCDC, 0x12);
```

9.3.2 Target voltage adjustment

To adjust target voltage of VDD1P8 and VDD1P5. The following code could be used:

```
DCDC_AdjustTargetVoltage(DCDC, vdd1p45Boost, vdd1p45Buck, vdd1p8); // Target
       voltage is unlocked.
// DCDC_STS_DC_OK bit will be de-asserted after target register changes.
// After output voltage settling to new target value, DCDC_STS_DC_OK will be asserted.
while (kDCDC_LockedOKStatus == (kDCDC_LockedOKStatus &
      DCDC_GetStatusFlags(DCDC)))
{
}
```

9.3.3 Pulsed mode

Before entering pulsed mode, the target voltage should be locked. Also, there are some recommended setting.

```
dcdc_low_power_config_t dcdcLpConfigStruct;
dcdc_loop_control_config_t dcdcLoopCtrlConfigStruct;

DCDC_GetDefaultLoopControlConfig(&dcdcLoopCtrlConfigStruct);
dcdcLoopCtrlConfigStruct.enableDiffHysteresis = true; //DCDC_LOOPCTRL_EN_DF_HYST = 1
dcdcLoopCtrlConfigStruct.enableCommonHysteresis = true; //DCDC_LOOPCTRL_EN_CM_HYST =
       1
dcdcLoopCtrlConfigStruct.enableInvertHysteresisSign = true; //
      DCDC_LOOPCTRL_HYST_SIGN = 1
DCDC_SetLoopControlConfig(DCDC, &dcdcLoopCtrlConfigStruct);

DCDC_GetDefaultLowPowerConfig(&dcdcLpConfigStruct);
dcdcLpConfigStruct.enableDiffComparators = true;
DCDC_SetLowPowerConfig(DCDC, &dcdcLpConfigStruct)

// Enter low power mode.
```

Data Structures

- struct dcdc_low_power_config_t
  Configuration for the low power. More...
- struct dcdc_loop_control_config_t
  Configuration for the loop control. More...
- struct dcdc_min_power_config_t
  Configuration for min power setting. More...
- struct dcdc_pulsed_integrator_config_t
  Configuration for the integrator in pulsed mode. More...

Macros

- #define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  DCDC driver version.

Enumerations

- enum _dcdc_status_flags_t {
  kDCDC_LockedOKStatus = (1U << 0),
  kDCDC_PSwitchStatus = (1U << 1),
  kDCDC_PSwitchInterruptStatus = (1U << 2) }
  Status flags.
- enum _dcdc_interrupt_enable_t { kDCDC_PSwitchEdgeDetectInterruptEnable = DCDC_REG6_-PSWITCH_INT_MUTE_MASK }
  Interrupts.
- enum _dcdc_pswitch_detect_event_t {
  kDCDC_PSwitchFallingEdgeDetectEnable = DCDC_REG6_PSWITCH_INT_FALL_EN_MASK,
  kDCDC_PSwitchRisingEdgeDetectEnable = DCDC_REG6_PSWITCH_INT_RISE_EN_MASK }
  Events for PSWITCH signal(pin).
- enum dcdc_work_mode_t {
  kDCDC_WorkInContinuousMode = 0U,
  kDCDC_WorkInPulsedMode = 1U }
  DCDC work mode in SoC's low power condition.
- enum dcdc_hysteretic_threshold_offset_value_t {
  kDCDC_HystereticThresholdOffset0mV = 0U,
  kDCDC_HystereticThresholdOffset25mV = 1U,
  kDCDC_HystereticThresholdOffset50mV = 2U,
  kDCDC_HystereticThresholdOffset75mV = 3U }
  Hysteretic upper/lower threshold value in low power mode.
- enum dcdc_vbat_divider_t {
  kDCDC_VBatVoltageDividerOff = 0U,
  kDCDC_VBatVoltageDivider1 = 1U,
  kDCDC_VBatVoltageDivider2 = 2U,
  kDCDC_VBatVoltageDivider4 = 3U }
  VBAT voltage divider.
- enum dcdc_clock_source_t { kDCDC_ClockAutoSwitch = 0U }
  Oscillator clock option.

Initialization and deinitialization

- void DCDC_Init (DCDC_Type ∗base)
  Enable the access to DCDC registers.
- void DCDC_Deinit (DCDC_Type ∗base)
  Disable the access to DCDC registers.

Status

- uint32_t DCDC_GetStatusFlags (DCDC_Type ∗base)
  Get status flags.
- void DCDC_ClearStatusFlags (DCDC_Type ∗base, uint32_t mask)
  Clear status flags.

Interrupts

- static void DCDC_EnableInterrupts (DCDC_Type ∗base, uint32_t mask)
  Enable interrupts.
- static void DCDC_DisableInterrupts (DCDC_Type ∗base, uint32_t mask)
  Disable interrupts.
- void DCDC_SetPSwitchInterruptConfig (DCDC_Type ∗base, uint32_t mask)
  Configure the PSWITCH interrupts.

Misc control.

- void DCDC_GetDefaultLowPowerConfig (dcdc_low_power_config_t ∗config)
  Get the default setting for low power configuration.
- void DCDC_SetLowPowerConfig (DCDC_Type ∗base, const dcdc_low_power_config_t ∗config)
  Configure the low power for DCDC.
- void DCDC_GetDefaultLoopControlConfig (dcdc_loop_control_config_t ∗config)
  Get the default setting for loop control configuration.
- void DCDC_SetLoopControlConfig (DCDC_Type ∗base, const dcdc_loop_control_config_t ∗config)
  Configure the loop control for DCDC.
- static void DCDC_EnableXtalOKDetectionCircuit (DCDC_Type ∗base, bool enable)
  Enable the XTAL OK detection circuit.
- static void DCDC_EnableOutputRangeComparator (DCDC_Type ∗base, bool enable)
  Enable the output range comparator.
- static void DCDC_EnableReduceCurrent (DCDC_Type ∗base, bool enable)
  Enable to reduce the DCDC current.
- void DCDC_SetClockSource (DCDC_Type ∗base, dcdc_clock_source_t clockSource)
  Set the clock source for DCDC.
- static void DCDC_SetBatteryVoltageDivider (DCDC_Type ∗base, dcdc_vbat_divider_t divider)
  Set the battery voltage divider for ADC sample.
- void DCDC_SetBatteryMonitorValue (DCDC_Type ∗base, uint32_t battValue)
  Set battery monitor value.
- static void DCDC_DoSoftShutdown (DCDC_Type ∗base)
  Software shutdown the DCDC module to stop the power supply for chip.
- static void DCDC_SetUpperLimitDutyCycleBoost (DCDC_Type ∗base, uint32_t value)
  Set upper limit duty cycle limit in DCDC converter in Boost mode.
- static void DCDC_SetUpperLimitDutyCycleBuck (DCDC_Type ∗base, uint32_t value)

Set upper limit duty cycle limit in DCDC converter in Buck mode.
- static void DCDC_AdjustDutyCycleSwitchingTargetOutput (DCDC_Type *base, uint32_t value)
  Adjust value of duty cycle when switching between VDD1P45 and VDD1P8.
- static void DCDC_LockTargetVoltage (DCDC_Type *base)
  Lock the setting of target voltage.
- void DCDC_AdjustTargetVoltage (DCDC_Type *base, uint32_t vdd1p5xBoost, uint32_t vdd1p5xBuck, uint32_t vdd1p8)
  Adjust the target voltage of DCDC output.
- void DCDC_GetDefaultMinPowerDefault (dcdc_min_power_config_t *config)
  Get the default configuration for min power.
- void DCDC_SetMinPowerConfig (DCDC_Type *base, const dcdc_min_power_config_t *config)
  Configure for the min power.
- void DCDC_GetDefaultPulsedIntegratorConfig (dcdc_pulsed_integrator_config_t *config)
  Get the default setting for integrator configuration in pulsed mode.
- void DCDC_SetPulsedIntegratorConfig (DCDC_Type *base, const dcdc_pulsed_integrator_config_t *config)
  Configure the integrator in pulsed mode.

9.4 Data Structure Documentation

9.4.1 struct dcdc_low_power_config_t

Data Fields

- dcdc_work_mode_t workModeInVLPRW
  Select the behavior of DCDC in device VLPR and VLPW low power modes.
- dcdc_work_mode_t workModeInVLPS
  Select the behavior of DCDC in device VLPS low power modes.
- bool enableHysteresisVoltageSense
  Enable hysteresis in low power voltage sense.
- bool enableAdjustHystereticValueSense
  Adjust hysteretic value in low power voltage sense.
- bool enableHystersisComparator
  Enable hysteresis in low power comparator.
- bool enableAdjustHystereticValueComparator
  Adjust hysteretic value in low power comparator.
- dcdc_hysteretic_threshold_offset_value_t hystereticUpperThresholdValue
  Configure the hysteretic upper threshold value in low power mode.
- dcdc_hysteretic_threshold_offset_value_t hystereticLowerThresholdValue
  Configure the hysteretic lower threshold value in low power mode.
- bool enableDiffComparators
  Enable low power differential comparators, to sense lower supply in pulsed mode.

9.4.1.0.0.9 Field Documentation

9.4.1.0.0.9.1 dcdc_work_mode_t dcdc_low_power_config_t::workModeInVLPRW

9.4.1.0.0.9.2 dcdc_work_mode_t dcdc_low_power_config_t::workModeInVLPS

9.4.1.0.0.9.3 bool dcdc_low_power_config_t::enableHysteresisVoltageSense

9.4.1.0.0.9.4 bool dcdc_low_power_config_t::enableAdjustHystereticValueSense

9.4.1.0.0.9.5 bool dcdc_low_power_config_t::enableHystersisComparator

9.4.1.0.0.9.6 bool dcdc_low_power_config_t::enableAdjustHystereticValueComparator

9.4.1.0.0.9.7 dcdc_hysteretic_threshold_offset_value_t dcdc_low_power_config_t::hysteretic-UpperThresholdValue

9.4.1.0.0.9.8 dcdc_hysteretic_threshold_offset_value_t dcdc_low_power_config_t::hysteretic-LowerThresholdValue

9.4.1.0.0.9.9 bool dcdc_low_power_config_t::enableDiffComparators

9.4.2 struct dcdc_loop_control_config_t

Data Fields

- bool enableDiffHysteresis
  Enable hysteresis in switching converter differential mode analog comparators.
- bool enableCommonHysteresis
  Enable hysteresis in switching converter common mode analog comparators.
- bool enableDiffHysteresisThresh
  This field act the same rule as enableDiffHysteresis.
- bool enableCommonHysteresisThresh
  This field act the same rule as enableCommonHysteresis.
- bool enableInvertHysteresisSign
  Invert the sign of the hysteresis in DC-DC analog comparators.

9.4.2.0.0.10 Field Documentation

9.4.2.0.0.10.1 bool dcdc_loop_control_config_t::enableDiffHysteresis

This feature improves transient supply ripple and efficiency.

9.4.2.0.0.10.2 bool dcdc_loop_control_config_t::enableCommonHysteresis

This feature improves transient supply ripple and efficiency.

9.4.2.0.0.10.3 bool dcdc_loop_control_config_t::enableDiffHysteresisThresh

However, if this field is enabled along with the enableDiffHysteresis, the Hysteresis would be doubled.

9.4.2.0.0.10.4 bool dcdc_loop_control_config_t::enableCommonHysteresisThresh

However, if this field is enabled along with the enableCommonHysteresis, the Hysteresis would be doubled.

9.4.2.0.0.10.5 bool dcdc_loop_control_config_t::enableInvertHysteresisSign

9.4.3 struct dcdc_min_power_config_t

Data Fields

- bool enableUseHalfFetForContinuous
  Use half switch FET for the continuous mode.
- bool enableUseDoubleFetForContinuous
  Use double switch FET for the continuous mode.
- bool enableUseHalfFreqForContinuous
  Set DCDC clock to half frequency for the continuous mode.
- bool enableUseHalfFetForPulsed
  Use half switch FET for the Pulsed mode.
- bool enableUseDoubleFetForPulsed
  Use double switch FET for the Pulsed mode.
- bool enableUseHalfFreqForPulsed
  Set DCDC clock to half frequency for the Pulsed mode.

9.4.3.0.0.11 Field Documentation

9.4.3.0.0.11.1 bool dcdc_min_power_config_t::enableUseHalfFetForContinuous

9.4.3.0.0.11.2 bool dcdc_min_power_config_t::enableUseDoubleFetForContinuous

9.4.3.0.0.11.3 bool dcdc_min_power_config_t::enableUseHalfFreqForContinuous

9.4.3.0.0.11.4 bool dcdc_min_power_config_t::enableUseHalfFetForPulsed

9.4.3.0.0.11.5 bool dcdc_min_power_config_t::enableUseDoubleFetForPulsed

9.4.3.0.0.11.6 bool dcdc_min_power_config_t::enableUseHalfFreqForPulsed

9.4.4 struct dcdc_pulsed_integrator_config_t

Data Fields

- bool enableUseUserIntegratorValue
  Enable to use the setting value in userIntegratorValue field.
- uint32_t userIntegratorValue
  User defined integrator value.
- bool enablePulseRunSpeedup
  Enable pulse run speedup.

9.4.4.0.0.12 Field Documentation

9.4.4.0.0.12.1 bool dcdc_pulsed_integrator_config_t::enableUseUserIntegratorValue

Otherwise, the predefined hardware setting would be applied internally.

9.4.4.0.0.12.2 uint32_t dcdc_pulsed_integrator_config_t::userIntegratorValue

The available value is 19-bit.

9.4.4.0.0.12.3 bool dcdc_pulsed_integrator_config_t::enablePulseRunSpeedup

9.5 Macro Definition Documentation

9.5.1 #define FSL_DCDC_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Version 2.0.1.

9.6 Enumeration Type Documentation

9.6.1 enum _dcdc_status_flags_t

Enumerator

kDCDC_LockedOKStatus Status to indicate DCDC lock. Read only bit.
kDCDC_PSwitchStatus Status to indicate PSWITCH signal. Read only bit.
kDCDC_PSwitchInterruptStatus PSWITCH edge detection interrupt status.

9.6.2 enum _dcdc_interrupt_enable_t

Enumerator

kDCDC_PSwitchEdgeDetectInterruptEnable Enable the edge detect interrupt.

9.6.3 enum _dcdc_pswitch_detect_event_t

Enumerator

kDCDC_PSwitchFallingEdgeDetectEnable Enable falling edge detect.
kDCDC_PSwitchRisingEdgeDetectEnable Enable rising edge detect.

9.6.4 enum dcdc_work_mode_t

Enumerator

kDCDC_WorkInContinuousMode DCDC works in continuous mode when SOC is in low power mode.
kDCDC_WorkInPulsedMode DCDC works in pulsed mode when SOC is in low power mode.

9.6.5 enum dcdc_hysteretic_threshold_offset_value_t

Enumerator

kDCDC_HystereticThresholdOffset0mV Target voltage value +/- 0mV.
kDCDC_HystereticThresholdOffset25mV Target voltage value +/- 25mV.
kDCDC_HystereticThresholdOffset50mV Target voltage value +/- 50mV.
kDCDC_HystereticThresholdOffset75mV Target voltage value +/- 75mV.

9.6.6 enum dcdc_vbat_divider_t

Enumerator

kDCDC_VBatVoltageDividerOff The sensor signal is disabled.
kDCDC_VBatVoltageDivider1 VBat.
kDCDC_VBatVoltageDivider2 VBat/2.
kDCDC_VBatVoltageDivider4 VBat/4.

9.6.7 enum dcdc_clock_source_t

Enumerator

kDCDC_ClockAutoSwitch Automatic clock switch from internal oscillator to external clock.

9.7 Function Documentation

9.7.1 void DCDC_Init (DCDC_Type * *base*)

Parameters

base	DCDC peripheral base address.

9.7.2 void DCDC_Deinit (DCDC_Type * *base*)

Parameters

base	DCDC peripheral base address.

9.7.3 uint32_t DCDC_GetStatusFlags (DCDC_Type * *base*)

base DCDC peripheral base address.

Returns

Masks of asserted status flags. See to "_dcdc_status_flags_t".

9.7.4 void DCDC_ClearStatusFlags (DCDC_Type * *base,* uint32_t *mask*)

base DCDC peripheral base address. mask Mask of status values that would be cleared. See to "_dcdc_-status_flags_t".

9.7.5 static void DCDC_EnableInterrupts (DCDC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	DCDC peripheral base address.
mask	Mask of interrupt events that would be enabled. See to "_dcdc_interrupt_enable_t".

9.7.6 static void DCDC_DisableInterrupts (DCDC_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	DCDC peripheral base address.
mask	Mask of interrupt events that would be disabled. See to "_dcdc_interrupt_enable_t".

9.7.7 void DCDC_SetPSwitchInterruptConfig (DCDC_Type * *base*, uint32_t *mask*)

There are PSWITCH interrupt events can be triggered by falling edge or rising edge. So user can set the interrupt events that would be triggered with this function. Un-asserted events would be disabled. The interrupt of PSwitch should be enabled as well if to sense the PSWTICH event. By default, no interrupt events would be enabled.

Parameters

base	DCDC peripheral base address.
mask	Mask of interrupt events for PSwtich. See to "_dcdc_pswitch_detect_event_t".

9.7.8 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->workModeInVLPRW = kDCDC_WorkInPulsedMode;
*   config->workModeInVLPS = kDCDC_WorkInPulsedMode;
*   config->enableHysteresisVoltageSense = true;
*   config->enableAdjustHystereticValueSense = false;
*   config->enableHystersisComparator = true;
*   config->enableAdjustHystereticValueComparator = false;
*   config->hystereticUpperThresholdValue = kDCDC_HystereticThresholdOffset75mV
      ;
*   config->hystereticLowerThresholdValue = kDCDC_HystereticThresholdOffset0mV
      ;
*   config->enableDiffComparators = false;
*
```

Parameters

config	Pointer to configuration structure. See to "dcdc_low_power_config_t".

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

Parameters

base	DCDC peripheral base address.
config	Pointer to configuration structure. See to "dcdc_low_power_config_t".

9.7.10 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->enableDiffHysteresis = false;
*   config->enableCommonHysteresis = false;
*   config->enableDiffHysteresisThresh = false;
*   config->enableCommonHysteresisThresh = false;
*   config->enableInvertHysteresisSign = false;
*
```

Parameters

config	Pointer to configuration structure. See to "dcdc_loop_control_config_t".

9.7.11 void DCDC_SetLoopControlConfig (DCDC_Type * *base,* const dcdc_loop_control_config_t * *config*)

Parameters

base	DCDC peripheral base address.
config	Pointer to configuration structure. See to "dcdc_loop_control_config_t".

9.7.12 static void DCDC_EnableXtalOKDetectionCircuit (DCDC_Type * *base,* bool *enable*) [inline], [static]

The XTAL OK detection circuit is enabled by default.

Parameters

base	DCDC peripheral base address.
enable	Enable the feature or not.

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

The output range comparator is enabled by default.

Parameters

base	DCDC peripheral base address.
enable	Enable the feature or not.

9.7.14 static void DCDC_EnableReduceCurrent (DCDC_Type * *base*, bool *enable*) [inline], [static]

To enable this feature will save approximately 20 µA in RUN mode. This feature is disabled by default.

Parameters

base	DCDC peripheral base address.
enable	Enable the feature or not.

9.7.15 void DCDC_SetClockSource (DCDC_Type * *base*, dcdc_clock_source_t *clockSource*)

This function is to set the clock source for DCDC. By default, DCDC can switch the clock from internal oscillator to external clock automatically. Once the application chooses to use the external clock with function, the internal oscillator would be powered down. However, the internal oscillator could be powered down only when 32MHz crystal oscillator is available.

Parameters

base	DCDC peripheral base address.

clockSource	Clock source for DCDC. See to "dcdc_clock_source_t".

9.7.16 static void DCDC_SetBatteryVoltageDivider (DCDC_Type * *base,* dcdc_vbat_divider_t *divider*) [inline], [static]

This function controls VBAT voltage divider. The divided VBAT output is input to an ADC channel which allows the battery voltage to be measured.

Parameters

base	DCDC peripheral base address.
divider	Setting divider selection. See to "dcdc_vbat_divider_t"

9.7.17 void DCDC_SetBatteryMonitorValue (DCDC_Type * *base,* uint32_t *battValue*)

This function is to set the battery monitor value. If the feature of monitoring battery voltage is enabled (with non-zero value set), user should set the battery voltage measured with an 8 mV LSB resolution from the ADC sample channel. It would improve efficiency and minimize ripple.

Parameters

base	DCDC peripheral base address.
battValue	Battery voltage measured with an 8 mV LSB resolution with 10-bit ADC sample. Setting 0x0 would disable feature of monitoring battery voltage.

9.7.18 static void DCDC_DoSoftShutdown (DCDC_Type * *base*) [inline], [static]

This function is to shutdown the DCDC module and stop the power supply for chip. Shutdown is only allowed in Buck mode, Manual start and PSWITCH must be at logic zero.

Parameters

base	DCDC peripheral base address.

9.7.19 static void DCDC_SetUpperLimitDutyCycleBoost (DCDC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DCDC peripheral base address.
value	Setting value for limit duty cycle. Available range is 0-127.

9.7.20 static void DCDC_SetUpperLimitDutyCycleBuck (DCDC_Type * *base,* uint32_t *value*) [inline], [static]

Parameters

base	DCDC peripheral base address.
value	Setting value for limit duty cycle. Available range is 0-127.

9.7.21 static void DCDC_AdjustDutyCycleSwitchingTargetOutput (DCDC_Type * *base,* uint32_t *value*) [inline], [static]

Adjust value of duty cycle when switching between VDD1P45 and VDD1P8. The unit is 1/32 or 3.125%.

Parameters

base	DCDC peripheral base address.
value	Setting adjust value. The available range is 0-15. The unit is 1/32 or 3.125&.

9.7.22 static void DCDC_LockTargetVoltage (DCDC_Type * *base*) [inline], [static]

This function is to lock the setting of target voltage. This function should be called before entering the low power modes to lock the target voltage.

Parameters

base	DCDC peripheral base address.

9.7.23 void DCDC_AdjustTargetVoltage (DCDC_Type * *base,* uint32_t *vdd1p5xBoost,* uint32_t *vdd1p5xBuck,* uint32_t *vdd1p8*)

This function is to adjust the target voltage of DCDC output. It would unlock the setting of target voltages, change them and finally wait until the output is stabled.

Parameters

base	DCDC peripheral base address.
vdd1p5xBoost	Target value of VDD1P5X in boost mode, 25 mV each step from 0x00 to 0x0F. 0x00 is for 1.275V.
vdd1p5xBuck	Target value of VDD1P5X in buck mode, 25 mV each step from 0x00 to 0x0F. 0x00 is for 1.275V.
vdd1p8	Target value of VDD1P8, 25 mV each step in two ranges, from 0x00 to 0x11 and 0x20 to 0x3F and 50 mV each step in range from 0x12 to 0x1F. 0x00 is for 1.65V, 0x20 is for 2.8V.

9.7.24 void DCDC_GetDefaultMinPowerDefault (dcdc_min_power_config_t * *config*)

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

```
*   config->enableUseHalfFetForContinuous = false;
*   config->enableUseDoubleFetForContinuous = false;
*   config->enableUseHalfFreqForContinuous = false;
*   config->enableUseHalfFetForPulsed = false;
*   config->enableUseDoubleFetForPulsed = false;
*   config->enableUseHalfFreqForPulsed = false;
*
```

Parameters

config	Pointer to configuration structure. See to "dcdc_min_power_config_t".

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

Parameters

base	DCDC peripheral base address.
config	Pointer to configuration structure. See to "dcdc_min_power_config_t".

9.7.26 void DCDC_GetDefaultPulsedIntegratorConfig (dcdc_pulsed_integrator_-config_t ∗ *config*)

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

```
*   config->enableUseUserIntegratorValue = false;
*   config->userIntegratorValue = 0U;
*   config->enablePulseRunSpeedup = false;
*
```

Parameters

config	Pointer to configuration structure. See to "dcdc_pulsed_integrator_config_t".

9.7.27 void DCDC_SetPulsedIntegratorConfig (DCDC_Type ∗ *base,* const dcdc_pulsed_integrator_config_t ∗ *config*)

Parameters

base	DCDC peripheral base address. Pointer to configuration structure. See to "dcdc_-pulsed_integrator_config_t".

Chapter 10
DMAMUX: Direct Memory Access Multiplexer Driver

10.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access Multiplexer (DMAMUX) of MCUXpresso SDK devices.

10.2 Typical use case

10.2.1 DMAMUX Operation

```
DMAMUX_Init(DMAMUX0);
DMAMUX_SetSource(DMAMUX0, channel, source);
DMAMUX_EnableChannel(DMAMUX0, channel);
...
DMAMUX_DisableChannel(DMAMUX, channel);
DMAMUX_Deinit(DMAMUX0);
```

Driver version

- #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  DMAMUX driver version 2.0.2.

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.

10.3 Macro Definition Documentation

10.3.1 #define FSL_DMAMUX_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

10.4 Function Documentation

10.4.1 void DMAMUX_Init (DMAMUX_Type * *base*)

This function ungates the DMAMUX clock.

Parameters

base	DMAMUX peripheral base address.

10.4.2 void DMAMUX_Deinit (DMAMUX_Type ∗ *base*)

This function gates the DMAMUX clock.

Parameters

base	DMAMUX peripheral base address.

10.4.3 static void DMAMUX_EnableChannel (DMAMUX_Type ∗ *base,* uint32_t *channel*) [inline], [static]

This function enables the DMAMUX channel.

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.

10.4.4 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

base	DMAMUX peripheral base address.

channel	DMAMUX channel number.

10.4.5 static void DMAMUX_SetSource (DMAMUX_Type * *base,* uint32_t *channel,* uint32_t *source*) [inline], [static]

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.
source	Channel source, which is used to trigger the DMA transfer.

10.4.6 static void DMAMUX_EnablePeriodTrigger (DMAMUX_Type * *base,* uint32_t *channel*) [inline], [static]

This function enables the DMAMUX period trigger feature.

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.

10.4.7 static void DMAMUX_DisablePeriodTrigger (DMAMUX_Type * *base,* uint32_t *channel*) [inline], [static]

This function disables the DMAMUX period trigger.

Parameters

base	DMAMUX peripheral base address.
channel	DMAMUX channel number.

Chapter 11
DSPI: Serial Peripheral Interface Driver

11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Serial Peripheral Interface (SPI) module of MCUXpresso SDK devices.

Modules

- DSPI DMA Driver
- DSPI Driver
- DSPI FreeRTOS Driver
- DSPI eDMA Driver

11.2 DSPI Driver

11.2.1 Overview

This section describes the programming interface of the DSPI Peripheral driver. The DSPI driver configures the DSPI module and provides the functional and transactional interfaces to build the DSPI application.

11.2.2 Typical use case

11.2.2.1 Master Operation

```
dspi_master_handle_t g_m_handle; //global variable
dspi_master_config_t  masterConfig;
masterConfig.whichCtar                                = kDSPI_Ctar0;
masterConfig.ctarConfig.baudRate                      = baudrate;
masterConfig.ctarConfig.bitsPerFrame                  = 8;
masterConfig.ctarConfig.cpol                          =
      kDSPI_ClockPolarityActiveHigh;
masterConfig.ctarConfig.cpha                          =
      kDSPI_ClockPhaseFirstEdge;
masterConfig.ctarConfig.direction                     =
      kDSPI_MsbFirst;
masterConfig.ctarConfig.pcsToSckDelayInNanoSec        = 1000000000 /
      baudrate ;
masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec    = 1000000000 /
      baudrate ;
masterConfig.ctarConfig.betweenTransferDelayInNanoSec = 1000000000 /
       baudrate ;
masterConfig.whichPcs                                 = kDSPI_Pcs0;
masterConfig.pcsActiveHighOrLow                       =
      kDSPI_PcsActiveLow;
masterConfig.enableContinuousSCK                      = false;
masterConfig.enableRxFifoOverWrite                    = false;
masterConfig.enableModifiedTimingFormat               = false;
masterConfig.samplePoint                              =
      kDSPI_SckToSin0Clock;
DSPI_MasterInit(base, &masterConfig, srcClock_Hz);

//srcClock_Hz = CLOCK_GetFreq(xxx);
DSPI_MasterInit(base, &masterConfig, srcClock_Hz);

DSPI_MasterTransferCreateHandle(base, &g_m_handle, NULL, NULL);

masterXfer.txData      = masterSendBuffer;
masterXfer.rxData      = masterReceiveBuffer;
masterXfer.dataSize    = transfer_dataSize;
masterXfer.configFlags = kDSPI_MasterCtar0 | kDSPI_MasterPcs0 ;
DSPI_MasterTransferBlocking(base, &g_m_handle, &masterXfer);
```

11.2.2.2 Slave Operation

```
dspi_slave_handle_t g_s_handle;//global variable
/*Slave config*/
slaveConfig.whichCtar                  = kDSPI_Ctar0;
slaveConfig.ctarConfig.bitsPerFrame    = 8;
slaveConfig.ctarConfig.cpol            = kDSPI_ClockPolarityActiveHigh;
slaveConfig.ctarConfig.cpha            = kDSPI_ClockPhaseFirstEdge;
```

```
slaveConfig.enableContinuousSCK        = false;
slaveConfig.enableRxFifoOverWrite      = false;
slaveConfig.enableModifiedTimingFormat = false;
slaveConfig.samplePoint                = kDSPI_SckToSin0Clock;
DSPI_SlaveInit(base, &slaveConfig);

slaveXfer.txData      = slaveSendBuffer0;
slaveXfer.rxData      = slaveReceiveBuffer0;
slaveXfer.dataSize    = transfer_dataSize;
slaveXfer.configFlags = kDSPI_SlaveCtar0;

bool isTransferCompleted = false;
DSPI_SlaveTransferCreateHandle(base, &g_s_handle, DSPI_SlaveUserCallback, &
      isTransferCompleted);

DSPI_SlaveTransferNonBlocking(&g_s_handle, &slaveXfer);


//void DSPI_SlaveUserCallback(SPI_Type *base, dspi_slave_handle_t *handle, status_t status, void
       *isTransferCompleted)
//{
//    if (status == kStatus_Success)
//    {
//        __NOP();
//    }
//    else if (status == kStatus_DSPI_Error)
//    {
//        __NOP();
//    }
//
//    *((bool *)isTransferCompleted) = true;
//
//    PRINTF("This is DSPI slave call back . \r\n");
//}
```

Data Structures

- struct dspi_command_data_config_t
  DSPI master command date configuration used for the SPIx_PUSHR. More...
- struct dspi_master_ctar_config_t
  DSPI master ctar configuration structure. More...
- struct dspi_master_config_t
  DSPI master configuration structure. More...
- struct dspi_slave_ctar_config_t
  DSPI slave ctar configuration structure. More...
- struct dspi_slave_config_t
  DSPI slave configuration structure. More...
- struct dspi_transfer_t
  DSPI master/slave transfer structure. More...
- struct dspi_master_handle_t
  DSPI master transfer handle structure used for transactional API. More...
- struct dspi_slave_handle_t
  DSPI slave transfer handle structure used for the transactional API. More...

Macros

- #define DSPI_DUMMY_DATA (0x00U)

DSPI dummy data if there is no Tx data.

- #define DSPI_MASTER_CTAR_SHIFT (0U)
  DSPI master CTAR shift macro; used internally.
- #define DSPI_MASTER_CTAR_MASK (0x0FU)
  DSPI master CTAR mask macro; used internally.
- #define DSPI_MASTER_PCS_SHIFT (4U)
  DSPI master PCS shift macro; used internally.
- #define DSPI_MASTER_PCS_MASK (0xF0U)
  DSPI master PCS mask macro; used internally.
- #define DSPI_SLAVE_CTAR_SHIFT (0U)
  DSPI slave CTAR shift macro; used internally.
- #define DSPI_SLAVE_CTAR_MASK (0x07U)
  DSPI slave CTAR mask macro; used internally.

Typedefs

- typedef void(∗ dspi_master_transfer_callback_t)(SPI_Type ∗base, dspi_master_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.
- typedef void(∗ dspi_slave_transfer_callback_t)(SPI_Type ∗base, dspi_slave_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.

Enumerations

- enum _dspi_status {
  kStatus_DSPI_Busy = MAKE_STATUS(kStatusGroup_DSPI, 0),
  kStatus_DSPI_Error = MAKE_STATUS(kStatusGroup_DSPI, 1),
  kStatus_DSPI_Idle = MAKE_STATUS(kStatusGroup_DSPI, 2),
  kStatus_DSPI_OutOfRange = MAKE_STATUS(kStatusGroup_DSPI, 3) }
  Status for the DSPI driver.
- enum _dspi_flags {
  kDSPI_TxCompleteFlag = SPI_SR_TCF_MASK,
  kDSPI_EndOfQueueFlag = SPI_SR_EOQF_MASK,
  kDSPI_TxFifoUnderflowFlag = SPI_SR_TFUF_MASK,
  kDSPI_TxFifoFillRequestFlag = SPI_SR_TFFF_MASK,
  kDSPI_RxFifoOverflowFlag = SPI_SR_RFOF_MASK,
  kDSPI_RxFifoDrainRequestFlag = SPI_SR_RFDF_MASK,
  kDSPI_TxAndRxStatusFlag = SPI_SR_TXRXS_MASK,
  kDSPI_AllStatusFlag }
  DSPI status flags in SPIx_SR register.
- enum _dspi_interrupt_enable {

kDSPI_TxCompleteInterruptEnable = SPI_RSER_TCF_RE_MASK,
kDSPI_EndOfQueueInterruptEnable = SPI_RSER_EOQF_RE_MASK,
kDSPI_TxFifoUnderflowInterruptEnable = SPI_RSER_TFUF_RE_MASK,
kDSPI_TxFifoFillRequestInterruptEnable = SPI_RSER_TFFF_RE_MASK,
kDSPI_RxFifoOverflowInterruptEnable = SPI_RSER_RFOF_RE_MASK,
kDSPI_RxFifoDrainRequestInterruptEnable = SPI_RSER_RFDF_RE_MASK,
kDSPI_AllInterruptEnable }

DSPI interrupt source.

- enum _dspi_dma_enable {
  kDSPI_TxDmaEnable = (SPI_RSER_TFFF_RE_MASK | SPI_RSER_TFFF_DIRS_MASK),
  kDSPI_RxDmaEnable = (SPI_RSER_RFDF_RE_MASK | SPI_RSER_RFDF_DIRS_MASK) }

  DSPI DMA source.
- enum dspi_master_slave_mode_t {
  kDSPI_Master = 1U,
  kDSPI_Slave = 0U }

  DSPI master or slave mode configuration.
- enum dspi_master_sample_point_t {
  kDSPI_SckToSin0Clock = 0U,
  kDSPI_SckToSin1Clock = 1U,
  kDSPI_SckToSin2Clock = 2U }

  DSPI Sample Point: Controls when the DSPI master samples SIN in the Modified Transfer Format.
- enum dspi_which_pcs_t {
  kDSPI_Pcs0 = 1U << 0,
  kDSPI_Pcs1 = 1U << 1,
  kDSPI_Pcs2 = 1U << 2,
  kDSPI_Pcs3 = 1U << 3,
  kDSPI_Pcs4 = 1U << 4,
  kDSPI_Pcs5 = 1U << 5 }

  DSPI Peripheral Chip Select (Pcs) configuration (which Pcs to configure).
- enum dspi_pcs_polarity_config_t {
  kDSPI_PcsActiveHigh = 0U,
  kDSPI_PcsActiveLow = 1U }

  DSPI Peripheral Chip Select (Pcs) Polarity configuration.
- enum _dspi_pcs_polarity {
  kDSPI_Pcs0ActiveLow = 1U << 0,
  kDSPI_Pcs1ActiveLow = 1U << 1,
  kDSPI_Pcs2ActiveLow = 1U << 2,
  kDSPI_Pcs3ActiveLow = 1U << 3,
  kDSPI_Pcs4ActiveLow = 1U << 4,
  kDSPI_Pcs5ActiveLow = 1U << 5,
  kDSPI_PcsAllActiveLow = 0xFFU }

  DSPI Peripheral Chip Select (Pcs) Polarity.
- enum dspi_clock_polarity_t {
  kDSPI_ClockPolarityActiveHigh = 0U,
  kDSPI_ClockPolarityActiveLow = 1U }

  DSPI clock polarity configuration for a given CTAR.
- enum dspi_clock_phase_t {

kDSPI_ClockPhaseFirstEdge = 0U,
kDSPI_ClockPhaseSecondEdge = 1U }

DSPI clock phase configuration for a given CTAR.

- enum dspi_shift_direction_t {
  kDSPI_MsbFirst = 0U,
  kDSPI_LsbFirst = 1U }

  DSPI data shifter direction options for a given CTAR.
- enum dspi_delay_type_t {
  kDSPI_PcsToSck = 1U,
  kDSPI_LastSckToPcs,
  kDSPI_BetweenTransfer }

  DSPI delay type selection.
- enum dspi_ctar_selection_t {
  kDSPI_Ctar0 = 0U,
  kDSPI_Ctar1 = 1U,
  kDSPI_Ctar2 = 2U,
  kDSPI_Ctar3 = 3U,
  kDSPI_Ctar4 = 4U,
  kDSPI_Ctar5 = 5U,
  kDSPI_Ctar6 = 6U,
  kDSPI_Ctar7 = 7U }

  DSPI Clock and Transfer Attributes Register (CTAR) selection.
- enum _dspi_transfer_config_flag_for_master {
  kDSPI_MasterCtar0 = 0U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar1 = 1U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar2 = 2U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar3 = 3U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar4 = 4U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar5 = 5U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar6 = 6U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterCtar7 = 7U << DSPI_MASTER_CTAR_SHIFT,
  kDSPI_MasterPcs0 = 0U << DSPI_MASTER_PCS_SHIFT,
  kDSPI_MasterPcs1 = 1U << DSPI_MASTER_PCS_SHIFT,
  kDSPI_MasterPcs2 = 2U << DSPI_MASTER_PCS_SHIFT,
  kDSPI_MasterPcs3 = 3U << DSPI_MASTER_PCS_SHIFT,
  kDSPI_MasterPcs4 = 4U << DSPI_MASTER_PCS_SHIFT,
  kDSPI_MasterPcs5 = 5U << DSPI_MASTER_PCS_SHIFT,
  kDSPI_MasterPcsContinuous = 1U << 20,
  kDSPI_MasterActiveAfterTransfer = 1U << 21 }

  Use this enumeration for the DSPI master transfer configFlags.
- enum _dspi_transfer_config_flag_for_slave { kDSPI_SlaveCtar0 = 0U << DSPI_SLAVE_CTAR-_SHIFT }

  Use this enumeration for the DSPI slave transfer configFlags.
- enum _dspi_transfer_state {
  kDSPI_Idle = 0x0U,
  kDSPI_Busy,

kDSPI_Error }
DSPI transfer state, which is used for DSPI transactional API state machine.

Driver version

- #define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))
  DSPI driver version 2.1.4.

Initialization and deinitialization

- void DSPI_MasterInit (SPI_Type *base, const dspi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Initializes the DSPI master.
- void DSPI_MasterGetDefaultConfig (dspi_master_config_t *masterConfig)
  Sets the dspi_master_config_t structure to default values.
- void DSPI_SlaveInit (SPI_Type *base, const dspi_slave_config_t *slaveConfig)
  DSPI slave configuration.
- void DSPI_SlaveGetDefaultConfig (dspi_slave_config_t *slaveConfig)
  Sets the dspi_slave_config_t structure to a default value.
- void DSPI_Deinit (SPI_Type *base)
  De-initializes the DSPI peripheral.
- static void DSPI_Enable (SPI_Type *base, bool enable)
  Enables the DSPI peripheral and sets the MCR MDIS to 0.

Status

- static uint32_t DSPI_GetStatusFlags (SPI_Type *base)
  Gets the DSPI status flag state.
- static void DSPI_ClearStatusFlags (SPI_Type *base, uint32_t statusFlags)
  Clears the DSPI status flag.

Interrupts

- void DSPI_EnableInterrupts (SPI_Type *base, uint32_t mask)
  Enables the DSPI interrupts.
- static void DSPI_DisableInterrupts (SPI_Type *base, uint32_t mask)
  Disables the DSPI interrupts.

DMA Control

- static void DSPI_EnableDMA (SPI_Type *base, uint32_t mask)
  Enables the DSPI DMA request.
- static void DSPI_DisableDMA (SPI_Type *base, uint32_t mask)
  Disables the DSPI DMA request.

- static uint32_t DSPI_MasterGetTxRegisterAddress (SPI_Type *base)
  Gets the DSPI master PUSHR data register address for the DMA operation.
- static uint32_t DSPI_SlaveGetTxRegisterAddress (SPI_Type *base)
  Gets the DSPI slave PUSHR data register address for the DMA operation.
- static uint32_t DSPI_GetRxRegisterAddress (SPI_Type *base)
  Gets the DSPI POPR data register address for the DMA operation.

Bus Operations

- static void DSPI_SetMasterSlaveMode (SPI_Type *base, dspi_master_slave_mode_t mode)
  Configures the DSPI for master or slave.
- static bool DSPI_IsMaster (SPI_Type *base)
  Returns whether the DSPI module is in master mode.
- static void DSPI_StartTransfer (SPI_Type *base)
  Starts the DSPI transfers and clears HALT bit in MCR.
- static void DSPI_StopTransfer (SPI_Type *base)
  Stops DSPI transfers and sets the HALT bit in MCR.
- static void DSPI_SetFifoEnable (SPI_Type *base, bool enableTxFifo, bool enableRxFifo)
  Enables or disables the DSPI FIFOs.
- static void DSPI_FlushFifo (SPI_Type *base, bool flushTxFifo, bool flushRxFifo)
  Flushes the DSPI FIFOs.
- static void DSPI_SetAllPcsPolarity (SPI_Type *base, uint32_t mask)
  Configures the DSPI peripheral chip select polarity simultaneously.
- uint32_t DSPI_MasterSetBaudRate (SPI_Type *base, dspi_ctar_selection_t whichCtar, uint32_-t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the DSPI baud rate in bits per second.
- void DSPI_MasterSetDelayScaler (SPI_Type *base, dspi_ctar_selection_t whichCtar, uint32_-t prescaler, uint32_t scaler, dspi_delay_type_t whichDelay)
  Manually configures the delay prescaler and scaler for a particular CTAR.
- uint32_t DSPI_MasterSetDelayTimes (SPI_Type *base, dspi_ctar_selection_t whichCtar, dspi_-delay_type_t whichDelay, uint32_t srcClock_Hz, uint32_t delayTimeInNanoSec)
  Calculates the delay prescaler and scaler based on the desired delay input in nanoseconds.
- static void DSPI_MasterWriteData (SPI_Type *base, dspi_command_data_config_t *command, uint16_t data)
  Writes data into the data buffer for master mode.
- void DSPI_GetDefaultDataCommandConfig (dspi_command_data_config_t *command)
  Sets the dspi_command_data_config_t structure to default values.
- void DSPI_MasterWriteDataBlocking (SPI_Type *base, dspi_command_data_config_t *command, uint16_t data)
  Writes data into the data buffer master mode and waits till complete to return.
- static uint32_t DSPI_MasterGetFormattedCommand (dspi_command_data_config_t *command)
  Returns the DSPI command word formatted to the PUSHR data register bit field.
- void DSPI_MasterWriteCommandDataBlocking (SPI_Type *base, uint32_t data)
  Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer master mode and waits till complete to return.
- static void DSPI_SlaveWriteData (SPI_Type *base, uint32_t data)
  Writes data into the data buffer in slave mode.
- void DSPI_SlaveWriteDataBlocking (SPI_Type *base, uint32_t data)
  Writes data into the data buffer in slave mode, waits till data was transmitted, and returns.

- static uint32_t DSPI_ReadData (SPI_Type *base)
  Reads data from the data buffer.

Transactional

- void DSPI_MasterTransferCreateHandle (SPI_Type *base, dspi_master_handle_t *handle, dspi_master_transfer_callback_t callback, void *userData)
  Initializes the DSPI master handle.
- status_t DSPI_MasterTransferBlocking (SPI_Type *base, dspi_transfer_t *transfer)
  DSPI master transfer data using polling.
- status_t DSPI_MasterTransferNonBlocking (SPI_Type *base, dspi_master_handle_t *handle, dspi_transfer_t *transfer)
  DSPI master transfer data using interrupts.
- status_t DSPI_MasterTransferGetCount (SPI_Type *base, dspi_master_handle_t *handle, size_t *count)
  Gets the master transfer count.
- void DSPI_MasterTransferAbort (SPI_Type *base, dspi_master_handle_t *handle)
  DSPI master aborts a transfer using an interrupt.
- void DSPI_MasterTransferHandleIRQ (SPI_Type *base, dspi_master_handle_t *handle)
  DSPI Master IRQ handler function.
- void DSPI_SlaveTransferCreateHandle (SPI_Type *base, dspi_slave_handle_t *handle, dspi_slave_transfer_callback_t callback, void *userData)
  Initializes the DSPI slave handle.
- status_t DSPI_SlaveTransferNonBlocking (SPI_Type *base, dspi_slave_handle_t *handle, dspi_transfer_t *transfer)
  DSPI slave transfers data using an interrupt.
- status_t DSPI_SlaveTransferGetCount (SPI_Type *base, dspi_slave_handle_t *handle, size_t *count)
  Gets the slave transfer count.
- void DSPI_SlaveTransferAbort (SPI_Type *base, dspi_slave_handle_t *handle)
  DSPI slave aborts a transfer using an interrupt.
- void DSPI_SlaveTransferHandleIRQ (SPI_Type *base, dspi_slave_handle_t *handle)
  DSPI Master IRQ handler function.

11.2.3 Data Structure Documentation

11.2.3.1 struct dspi_command_data_config_t

Data Fields

- bool isPcsContinuous
  Option to enable the continuous assertion of the chip select between transfers.
- dspi_ctar_selection_t whichCtar
  The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.
- dspi_which_pcs_t whichPcs
  The desired PCS signal to use for the data transfer.
- bool isEndOfQueue

Signals that the current transfer is the last in the queue.
- bool clearTransferCount
  Clears the SPI Transfer Counter (SPI_TCNT) before transmission starts.

11.2.3.1.0.13 Field Documentation

11.2.3.1.0.13.1 bool dspi_command_data_config_t::isPcsContinuous

11.2.3.1.0.13.2 dspi_ctar_selection_t dspi_command_data_config_t::whichCtar

11.2.3.1.0.13.3 dspi_which_pcs_t dspi_command_data_config_t::whichPcs

11.2.3.1.0.13.4 bool dspi_command_data_config_t::isEndOfQueue

11.2.3.1.0.13.5 bool dspi_command_data_config_t::clearTransferCount

11.2.3.2 struct dspi_master_ctar_config_t

Data Fields

- uint32_t baudRate
  Baud Rate for DSPI.
- uint32_t bitsPerFrame
  Bits per frame, minimum 4, maximum 16.
- dspi_clock_polarity_t cpol
  Clock polarity.
- dspi_clock_phase_t cpha
  Clock phase.
- dspi_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
  The last SCK to PCS delay time in nanoseconds; setting to 0 sets the minimum delay.
- uint32_t betweenTransferDelayInNanoSec
  `After the SCK delay time in nanoseconds; setting to 0 sets the minimum` *delay.*

11.2.3.2.0.14 Field Documentation

11.2.3.2.0.14.1 uint32_t dspi_master_ctar_config_t::baudRate

11.2.3.2.0.14.2 uint32_t dspi_master_ctar_config_t::bitsPerFrame

11.2.3.2.0.14.3 dspi_clock_polarity_t dspi_master_ctar_config_t::cpol

11.2.3.2.0.14.4 dspi_clock_phase_t dspi_master_ctar_config_t::cpha

11.2.3.2.0.14.5 dspi_shift_direction_t dspi_master_ctar_config_t::direction

11.2.3.2.0.14.6 uint32_t dspi_master_ctar_config_t::pcsToSckDelayInNanoSec

It also sets the boundary value if out of range.

11.2.3.2.0.14.7 uint32_t dspi_master_ctar_config_t::lastSckToPcsDelayInNanoSec

It also sets the boundary value if out of range.

11.2.3.2.0.14.8 uint32_t dspi_master_ctar_config_t::betweenTransferDelayInNanoSec

It also sets the boundary value if out of range.

11.2.3.3 struct dspi_master_config_t

Data Fields

- dspi_ctar_selection_t whichCtar
  The desired CTAR to use.
- dspi_master_ctar_config_t ctarConfig
  Set the ctarConfig to the desired CTAR.
- dspi_which_pcs_t whichPcs
  The desired Peripheral Chip Select (pcs).
- dspi_pcs_polarity_config_t pcsActiveHighOrLow
  The desired PCS active high or low.
- bool enableContinuousSCK
  CONT_SCKE, continuous SCK enable.
- bool enableRxFifoOverWrite
  ROOE, receive FIFO overflow overwrite enable.
- bool enableModifiedTimingFormat
  Enables a modified transfer format to be used if true.
- dspi_master_sample_point_t samplePoint
  Controls when the module master samples SIN in the Modified Transfer Format.

11.2.3.3.0.15 Field Documentation

11.2.3.3.0.15.1 dspi_ctar_selection_t dspi_master_config_t::whichCtar

11.2.3.3.0.15.2 dspi_master_ctar_config_t dspi_master_config_t::ctarConfig

11.2.3.3.0.15.3 dspi_which_pcs_t dspi_master_config_t::whichPcs

11.2.3.3.0.15.4 dspi_pcs_polarity_config_t dspi_master_config_t::pcsActiveHighOrLow

11.2.3.3.0.15.5 bool dspi_master_config_t::enableContinuousSCK

Note that the continuous SCK is only supported for CPHA = 1.

11.2.3.3.0.15.6 bool dspi_master_config_t::enableRxFifoOverWrite

If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

11.2.3.3.0.15.7 bool dspi_master_config_t::enableModifiedTimingFormat

11.2.3.3.0.15.8 dspi_master_sample_point_t dspi_master_config_t::samplePoint

It's valid only when CPHA=0.

11.2.3.4 struct dspi_slave_ctar_config_t

Data Fields

- uint32_t bitsPerFrame
  Bits per frame, minimum 4, maximum 16.
- dspi_clock_polarity_t cpol
  Clock polarity.
- dspi_clock_phase_t cpha
  Clock phase.

11.2.3.4.0.16 Field Documentation

11.2.3.4.0.16.1 uint32_t dspi_slave_ctar_config_t::bitsPerFrame

11.2.3.4.0.16.2 dspi_clock_polarity_t dspi_slave_ctar_config_t::cpol

11.2.3.4.0.16.3 dspi_clock_phase_t dspi_slave_ctar_config_t::cpha

Slave only supports MSB and does not support LSB.

11.2.3.5 struct dspi_slave_config_t

Data Fields

- dspi_ctar_selection_t whichCtar
  The desired CTAR to use.
- dspi_slave_ctar_config_t ctarConfig
  Set the ctarConfig to the desired CTAR.
- bool enableContinuousSCK
  CONT_SCKE, continuous SCK enable.
- bool enableRxFifoOverWrite
  ROOE, receive FIFO overflow overwrite enable.
- bool enableModifiedTimingFormat
  Enables a modified transfer format to be used if true.
- dspi_master_sample_point_t samplePoint
  Controls when the module master samples SIN in the Modified Transfer Format.

11.2.3.5.0.17 Field Documentation

11.2.3.5.0.17.1 dspi_ctar_selection_t dspi_slave_config_t::whichCtar

11.2.3.5.0.17.2 dspi_slave_ctar_config_t dspi_slave_config_t::ctarConfig

11.2.3.5.0.17.3 bool dspi_slave_config_t::enableContinuousSCK

Note that the continuous SCK is only supported for CPHA = 1.

11.2.3.5.0.17.4 bool dspi_slave_config_t::enableRxFifoOverWrite

If ROOE = 0, the incoming data is ignored and the data from the transfer that generated the overflow is also ignored. If ROOE = 1, the incoming data is shifted to the shift register.

11.2.3.5.0.17.5 bool dspi_slave_config_t::enableModifiedTimingFormat

11.2.3.5.0.17.6 dspi_master_sample_point_t dspi_slave_config_t::samplePoint

It's valid only when CPHA=0.

11.2.3.6 struct dspi_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; set from _dspi_transfer_config_flag_for_master if the transfer is

used for master or _dspi_transfer_config_flag_for_slave enumeration if the transfer is used for slave.

11.2.3.6.0.18 Field Documentation

11.2.3.6.0.18.1 uint8_t* dspi_transfer_t::txData

11.2.3.6.0.18.2 uint8_t* dspi_transfer_t::rxData

11.2.3.6.0.18.3 volatile size_t dspi_transfer_t::dataSize

11.2.3.6.0.18.4 uint32_t dspi_transfer_t::configFlags

11.2.3.7 struct _dspi_master_handle

Forward declaration of the _dspi_master_handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile uint32_t command
  The desired data command.
- volatile uint32_t lastCommand
  The desired last data command.
- uint8_t fifoSize
  FIFO dataSize.
- volatile bool isPcsActiveAfterTransfer
  Indicates whether the PCS signal is active after the last frame transfer.
- volatile bool isThereExtraByte
  Indicates whether there are extra bytes.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint8_t state
  DSPI transfer state, see _dspi_transfer_state.
- dspi_master_transfer_callback_t callback
  Completion callback.
- void * userData
  Callback user data.

11.2.3.7.0.19 Field Documentation

11.2.3.7.0.19.1 uint32_t dspi_master_handle_t::bitsPerFrame

11.2.3.7.0.19.2 volatile uint32_t dspi_master_handle_t::command

11.2.3.7.0.19.3 volatile uint32_t dspi_master_handle_t::lastCommand

11.2.3.7.0.19.4 uint8_t dspi_master_handle_t::fifoSize

11.2.3.7.0.19.5 volatile bool dspi_master_handle_t::isPcsActiveAfterTransfer

11.2.3.7.0.19.6 volatile bool dspi_master_handle_t::isThereExtraByte

11.2.3.7.0.19.7 uint8_t* volatile dspi_master_handle_t::txData

11.2.3.7.0.19.8 uint8_t* volatile dspi_master_handle_t::rxData

11.2.3.7.0.19.9 volatile size_t dspi_master_handle_t::remainingSendByteCount

11.2.3.7.0.19.10 volatile size_t dspi_master_handle_t::remainingReceiveByteCount

11.2.3.7.0.19.11 volatile uint8_t dspi_master_handle_t::state

11.2.3.7.0.19.12 dspi_master_transfer_callback_t dspi_master_handle_t::callback

11.2.3.7.0.19.13 void* dspi_master_handle_t::userData

11.2.3.8 struct _dspi_slave_handle

Forward declaration of the _dspi_slave_handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile bool isThereExtraByte
  Indicates whether there are extra bytes.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.
- volatile uint8_t state
  DSPI transfer state.

- volatile uint32_t errorCount
  Error count for slave transfer.
- dspi_slave_transfer_callback_t callback
  Completion callback.
- void ∗ userData
  Callback user data.

11.2.3.8.0.20 Field Documentation

11.2.3.8.0.20.1 uint32_t dspi_slave_handle_t::bitsPerFrame

11.2.3.8.0.20.2 volatile bool dspi_slave_handle_t::isThereExtraByte

11.2.3.8.0.20.3 uint8_t∗ volatile dspi_slave_handle_t::txData

11.2.3.8.0.20.4 uint8_t∗ volatile dspi_slave_handle_t::rxData

11.2.3.8.0.20.5 volatile size_t dspi_slave_handle_t::remainingSendByteCount

11.2.3.8.0.20.6 volatile size_t dspi_slave_handle_t::remainingReceiveByteCount

11.2.3.8.0.20.7 volatile uint8_t dspi_slave_handle_t::state

11.2.3.8.0.20.8 volatile uint32_t dspi_slave_handle_t::errorCount

11.2.3.8.0.20.9 dspi_slave_transfer_callback_t dspi_slave_handle_t::callback

11.2.3.8.0.20.10 void∗ dspi_slave_handle_t::userData

11.2.4 Macro Definition Documentation

11.2.4.1 #define FSL_DSPI_DRIVER_VERSION (MAKE_VERSION(2, 1, 4))

11.2.4.2 #define DSPI_DUMMY_DATA (0x00U)

Dummy data used for Tx if there is no txData.

11.2.4.3 #define DSPI_MASTER_CTAR_SHIFT (0U)

11.2.4.4 #define DSPI_MASTER_CTAR_MASK (0x0FU)

11.2.4.5 #define DSPI_MASTER_PCS_SHIFT (4U)

11.2.4.6 #define DSPI_MASTER_PCS_MASK (0xF0U)

11.2.4.7 #define DSPI_SLAVE_CTAR_SHIFT (0U)

11.2.4.8 #define DSPI_SLAVE_CTAR_MASK (0x07U)

11.2.5 Typedef Documentation

11.2.5.1 typedef void(∗ dspi_master_transfer_callback_t)(SPI_Type ∗base, dspi_master_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral address.
handle	Pointer to the handle for the DSPI master.
status	Success or error code describing whether the transfer completed.
userData	Arbitrary pointer-dataSized value passed from the application.

11.2.5.2 typedef void(∗ dspi_slave_transfer_callback_t)(SPI_Type ∗base, dspi_slave_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral address.
handle	Pointer to the handle for the DSPI slave.
status	Success or error code describing whether the transfer completed.
userData	Arbitrary pointer-dataSized value passed from the application.

11.2.6 Enumeration Type Documentation

11.2.6.1 enum _dspi_status

Enumerator

kStatus_DSPI_Busy DSPI transfer is busy.
kStatus_DSPI_Error DSPI driver error.
kStatus_DSPI_Idle DSPI is idle.
kStatus_DSPI_OutOfRange DSPI transfer out of range.

11.2.6.2 enum _dspi_flags

Enumerator

kDSPI_TxCompleteFlag Transfer Complete Flag.
kDSPI_EndOfQueueFlag End of Queue Flag.
kDSPI_TxFifoUnderflowFlag Transmit FIFO Underflow Flag.
kDSPI_TxFifoFillRequestFlag Transmit FIFO Fill Flag.
kDSPI_RxFifoOverflowFlag Receive FIFO Overflow Flag.
kDSPI_RxFifoDrainRequestFlag Receive FIFO Drain Flag.
kDSPI_TxAndRxStatusFlag The module is in Stopped/Running state.
kDSPI_AllStatusFlag All statuses above.

11.2.6.3 enum _dspi_interrupt_enable

Enumerator

kDSPI_TxCompleteInterruptEnable TCF interrupt enable.
kDSPI_EndOfQueueInterruptEnable EOQF interrupt enable.
kDSPI_TxFifoUnderflowInterruptEnable TFUF interrupt enable.
kDSPI_TxFifoFillRequestInterruptEnable TFFF interrupt enable, DMA disable.
kDSPI_RxFifoOverflowInterruptEnable RFOF interrupt enable.
kDSPI_RxFifoDrainRequestInterruptEnable RFDF interrupt enable, DMA disable.
kDSPI_AllInterruptEnable All above interrupts enable.

11.2.6.4 enum _dspi_dma_enable

Enumerator

kDSPI_TxDmaEnable TFFF flag generates DMA requests. No Tx interrupt request.
kDSPI_RxDmaEnable RFDF flag generates DMA requests. No Rx interrupt request.

11.2.6.5 enum dspi_master_slave_mode_t

Enumerator

kDSPI_Master DSPI peripheral operates in master mode.
kDSPI_Slave DSPI peripheral operates in slave mode.

11.2.6.6 enum dspi_master_sample_point_t

This field is valid only when the CPHA bit in the CTAR register is 0.

Enumerator

kDSPI_SckToSin0Clock 0 system clocks between SCK edge and SIN sample.
kDSPI_SckToSin1Clock 1 system clock between SCK edge and SIN sample.
kDSPI_SckToSin2Clock 2 system clocks between SCK edge and SIN sample.

11.2.6.7 enum dspi_which_pcs_t

Enumerator

kDSPI_Pcs0 Pcs[0].
kDSPI_Pcs1 Pcs[1].
kDSPI_Pcs2 Pcs[2].

kDSPI_Pcs3 Pcs[3].
kDSPI_Pcs4 Pcs[4].
kDSPI_Pcs5 Pcs[5].

11.2.6.8 enum dspi_pcs_polarity_config_t

Enumerator

kDSPI_PcsActiveHigh Pcs Active High (idles low).
kDSPI_PcsActiveLow Pcs Active Low (idles high).

11.2.6.9 enum _dspi_pcs_polarity

Enumerator

kDSPI_Pcs0ActiveLow Pcs0 Active Low (idles high).
kDSPI_Pcs1ActiveLow Pcs1 Active Low (idles high).
kDSPI_Pcs2ActiveLow Pcs2 Active Low (idles high).
kDSPI_Pcs3ActiveLow Pcs3 Active Low (idles high).
kDSPI_Pcs4ActiveLow Pcs4 Active Low (idles high).
kDSPI_Pcs5ActiveLow Pcs5 Active Low (idles high).
kDSPI_PcsAllActiveLow Pcs0 to Pcs5 Active Low (idles high).

11.2.6.10 enum dspi_clock_polarity_t

Enumerator

kDSPI_ClockPolarityActiveHigh CPOL=0. Active-high DSPI clock (idles low).
kDSPI_ClockPolarityActiveLow CPOL=1. Active-low DSPI clock (idles high).

11.2.6.11 enum dspi_clock_phase_t

Enumerator

kDSPI_ClockPhaseFirstEdge CPHA=0. Data is captured on the leading edge of the SCK and changed on the following edge.
kDSPI_ClockPhaseSecondEdge CPHA=1. Data is changed on the leading edge of the SCK and captured on the following edge.

11.2.6.12 enum dspi_shift_direction_t

Enumerator

kDSPI_MsbFirst Data transfers start with most significant bit.
kDSPI_LsbFirst Data transfers start with least significant bit. Shifting out of LSB is not supported for slave

11.2.6.13 enum dspi_delay_type_t

Enumerator

kDSPI_PcsToSck Pcs-to-SCK delay.
kDSPI_LastSckToPcs The last SCK edge to Pcs delay.
kDSPI_BetweenTransfer Delay between transfers.

11.2.6.14 enum dspi_ctar_selection_t

Enumerator

kDSPI_Ctar0 CTAR0 selection option for master or slave mode; note that CTAR0 and CTAR0_SLAVE are the same register address.
kDSPI_Ctar1 CTAR1 selection option for master mode only.
kDSPI_Ctar2 CTAR2 selection option for master mode only; note that some devices do not support CTAR2.
kDSPI_Ctar3 CTAR3 selection option for master mode only; note that some devices do not support CTAR3.
kDSPI_Ctar4 CTAR4 selection option for master mode only; note that some devices do not support CTAR4.
kDSPI_Ctar5 CTAR5 selection option for master mode only; note that some devices do not support CTAR5.
kDSPI_Ctar6 CTAR6 selection option for master mode only; note that some devices do not support CTAR6.
kDSPI_Ctar7 CTAR7 selection option for master mode only; note that some devices do not support CTAR7.

11.2.6.15 enum _dspi_transfer_config_flag_for_master

Enumerator

kDSPI_MasterCtar0 DSPI master transfer use CTAR0 setting.
kDSPI_MasterCtar1 DSPI master transfer use CTAR1 setting.
kDSPI_MasterCtar2 DSPI master transfer use CTAR2 setting.

kDSPI_MasterCtar3 DSPI master transfer use CTAR3 setting.
kDSPI_MasterCtar4 DSPI master transfer use CTAR4 setting.
kDSPI_MasterCtar5 DSPI master transfer use CTAR5 setting.
kDSPI_MasterCtar6 DSPI master transfer use CTAR6 setting.
kDSPI_MasterCtar7 DSPI master transfer use CTAR7 setting.
kDSPI_MasterPcs0 DSPI master transfer use PCS0 signal.
kDSPI_MasterPcs1 DSPI master transfer use PCS1 signal.
kDSPI_MasterPcs2 DSPI master transfer use PCS2 signal.
kDSPI_MasterPcs3 DSPI master transfer use PCS3 signal.
kDSPI_MasterPcs4 DSPI master transfer use PCS4 signal.
kDSPI_MasterPcs5 DSPI master transfer use PCS5 signal.
kDSPI_MasterPcsContinuous Indicates whether the PCS signal is continuous.
kDSPI_MasterActiveAfterTransfer Indicates whether the PCS signal is active after the last frame transfer.

11.2.6.16 enum _dspi_transfer_config_flag_for_slave

Enumerator

kDSPI_SlaveCtar0 DSPI slave transfer use CTAR0 setting. DSPI slave can only use PCS0.

11.2.6.17 enum _dspi_transfer_state

Enumerator

kDSPI_Idle Nothing in the transmitter/receiver.
kDSPI_Busy Transfer queue is not finished.
kDSPI_Error Transfer error.

11.2.7 Function Documentation

11.2.7.1 void DSPI_MasterInit (SPI_Type * *base*, const dspi_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

This function initializes the DSPI master configuration. This is an example use case.

```
*   dspi_master_config_t  masterConfig;
*   masterConfig.whichCtar                              = kDSPI_Ctar0;
*   masterConfig.ctarConfig.baudRate                    = 500000000U;
*   masterConfig.ctarConfig.bitsPerFrame                = 8;
*   masterConfig.ctarConfig.cpol                        =
      kDSPI_ClockPolarityActiveHigh;
*   masterConfig.ctarConfig.cpha                        =
      kDSPI_ClockPhaseFirstEdge;
*   masterConfig.ctarConfig.direction                   =
```

```
      kDSPI_MsbFirst;
*   masterConfig.ctarConfig.pcsToSckDelayInNanoSec        = 1000000000U /
      masterConfig.ctarConfig.baudRate ;
*   masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec    = 1000000000U
       / masterConfig.ctarConfig.baudRate ;
*   masterConfig.ctarConfig.betweenTransferDelayInNanoSec =
      1000000000U / masterConfig.ctarConfig.baudRate ;
*   masterConfig.whichPcs                                 = kDSPI_Pcs0;
*   masterConfig.pcsActiveHighOrLow                       =
      kDSPI_PcsActiveLow;
*   masterConfig.enableContinuousSCK                      = false;
*   masterConfig.enableRxFifoOverWrite                    = false;
*   masterConfig.enableModifiedTimingFormat               = false;
*   masterConfig.samplePoint                              =
      kDSPI_SckToSin0Clock;
*   DSPI_MasterInit(base, &masterConfig, srcClock_Hz);
*
```

Parameters

base	DSPI peripheral address.
masterConfig	Pointer to the structure dspi_master_config_t.
srcClock_Hz	Module source input clock in Hertz.

11.2.7.2 void DSPI_MasterGetDefaultConfig (dspi_master_config_t ∗ *masterConfig*)

The purpose of this API is to get the configuration structure initialized for the DSPI_MasterInit(). Users may use the initialized structure unchanged in the DSPI_MasterInit() or modify the structure before calling the DSPI_MasterInit(). Example:

```
*  dspi_master_config_t  masterConfig;
*  DSPI_MasterGetDefaultConfig(&masterConfig);
*
```

Parameters

masterConfig	pointer to dspi_master_config_t structure

11.2.7.3 void DSPI_SlaveInit (SPI_Type ∗ *base,* const dspi_slave_config_t ∗ *slaveConfig*)

This function initializes the DSPI slave configuration. This is an example use case.

```
*   dspi_slave_config_t  slaveConfig;
*  slaveConfig->whichCtar                 = kDSPI_Ctar0;
*  slaveConfig->ctarConfig.bitsPerFrame   = 8;
*  slaveConfig->ctarConfig.cpol           =
      kDSPI_ClockPolarityActiveHigh;
*  slaveConfig->ctarConfig.cpha           =
      kDSPI_ClockPhaseFirstEdge;
*  slaveConfig->enableContinuousSCK       = false;
```

```
*  slaveConfig->enableRxFifoOverWrite      = false;
*  slaveConfig->enableModifiedTimingFormat = false;
*  slaveConfig->samplePoint                = kDSPI_SckToSin0Clock;
*   DSPI_SlaveInit(base, &slaveConfig);
*
```

Parameters

base	DSPI peripheral address.
slaveConfig	Pointer to the structure dspi_master_config_t.

11.2.7.4 void DSPI_SlaveGetDefaultConfig (dspi_slave_config_t * *slaveConfig*)

The purpose of this API is to get the configuration structure initialized for the DSPI_SlaveInit(). Users may use the initialized structure unchanged in the DSPI_SlaveInit() or modify the structure before calling the DSPI_SlaveInit(). This is an example.

```
*  dspi_slave_config_t  slaveConfig;
*  DSPI_SlaveGetDefaultConfig(&slaveConfig);
*
```

Parameters

slaveConfig	Pointer to the dspi_slave_config_t structure.

11.2.7.5 void DSPI_Deinit (SPI_Type * *base*)

Call this API to disable the DSPI clock.

Parameters

base	DSPI peripheral address.

11.2.7.6 static void DSPI_Enable (SPI_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	DSPI peripheral address.
enable	Pass true to enable module, false to disable module.

11.2.7.7 static uint32_t DSPI_GetStatusFlags (SPI_Type * *base*) [inline], [static]

Parameters

base	DSPI peripheral address.

Returns

DSPI status (in SR register).

11.2.7.8 static void DSPI_ClearStatusFlags (SPI_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 bit to clear. The list of status bits is defined in the dspi_status_and_interrupt_request_t. The function uses these bit positions in its algorithm to clear the desired flag state. This is an example.

```
* DSPI_ClearStatusFlags(base, kDSPI_TxCompleteFlag|
    kDSPI_EndOfQueueFlag);
*
```

Parameters

base	DSPI peripheral address.
statusFlags	The status flag used from the type dspi_flags.

< The status flags are cleared by writing 1 (w1c).

11.2.7.9 void DSPI_EnableInterrupts (SPI_Type * *base*, uint32_t *mask*)

This function configures the various interrupt masks of the DSPI. The parameters are a base and an interrupt mask. Note, for Tx Fill and Rx FIFO drain requests, enable the interrupt request and disable the DMA request.

```
* DSPI_EnableInterrupts(base,
    kDSPI_TxCompleteInterruptEnable |
    kDSPI_EndOfQueueInterruptEnable );
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum _dspi_interrupt_enable.

11.2.7.10 static void DSPI_DisableInterrupts (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

```
* DSPI_DisableInterrupts(base,
    kDSPI_TxCompleteInterruptEnable |
    kDSPI_EndOfQueueInterruptEnable );
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum _dspi_interrupt_enable.

11.2.7.11 static void DSPI_EnableDMA (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
* DSPI_EnableDMA(base, kDSPI_TxDmaEnable |
    kDSPI_RxDmaEnable);
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum dspi_dma_enable.

11.2.7.12 static void DSPI_DisableDMA (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

This function configures the Rx and Tx DMA mask of the DSPI. The parameters are a base and a DMA mask.

```
* SPI_DisableDMA(base, kDSPI_TxDmaEnable | kDSPI_RxDmaEnable);
*
```

Parameters

base	DSPI peripheral address.
mask	The interrupt mask; use the enum dspi_dma_enable.

11.2.7.13 static uint32_t DSPI_MasterGetTxRegisterAddress (SPI_Type * *base*) [inline], [static]

This function gets the DSPI master PUSHR data register address because this value is needed for the DMA operation.

Parameters

base	DSPI peripheral address.

Returns

The DSPI master PUSHR data register address.

11.2.7.14 static uint32_t DSPI_SlaveGetTxRegisterAddress (SPI_Type * *base*) [inline], [static]

This function gets the DSPI slave PUSHR data register address as this value is needed for the DMA operation.

Parameters

base	DSPI peripheral address.

Returns

The DSPI slave PUSHR data register address.

11.2.7.15 static uint32_t DSPI_GetRxRegisterAddress (SPI_Type * *base*) [inline], [static]

This function gets the DSPI POPR data register address as this value is needed for the DMA operation.

Parameters

base	DSPI peripheral address.

Returns

The DSPI POPR data register address.

11.2.7.16 static void DSPI_SetMasterSlaveMode (SPI_Type * *base*, dspi_master_slave_mode_t *mode*) [inline], [static]

Parameters

base	DSPI peripheral address.
mode	Mode setting (master or slave) of type dspi_master_slave_mode_t.

11.2.7.17 static bool DSPI_IsMaster (SPI_Type * *base*) [inline], [static]

Parameters

base	DSPI peripheral address.

Returns

Returns true if the module is in master mode or false if the module is in slave mode.

11.2.7.18 static void DSPI_StartTransfer (SPI_Type * *base*) [inline], [static]

This function sets the module to start data transfer in either master or slave mode.

Parameters

base	DSPI peripheral address.

11.2.7.19 static void DSPI_StopTransfer (SPI_Type * *base*) [inline], [static]

This function stops data transfers in either master or slave modes.

Parameters

base	DSPI peripheral address.

11.2.7.20 static void DSPI_SetFifoEnable (SPI_Type * *base*, bool *enableTxFifo*, bool *enableRxFifo*) [inline], [static]

This function allows the caller to disable/enable the Tx and Rx FIFOs independently. Note that to disable, pass in a logic 0 (false) for the particular FIFO configuration. To enable, pass in a logic 1 (true).

Parameters

base	DSPI peripheral address.
enableTxFifo	Disables (false) the TX FIFO; Otherwise, enables (true) the TX FIFO
enableRxFifo	Disables (false) the RX FIFO; Otherwise, enables (true) the RX FIFO

11.2.7.21 static void DSPI_FlushFifo (SPI_Type * *base*, bool *flushTxFifo*, bool *flushRxFifo*) [inline], [static]

Parameters

base	DSPI peripheral address.
flushTxFifo	Flushes (true) the Tx FIFO; Otherwise, does not flush (false) the Tx FIFO
flushRxFifo	Flushes (true) the Rx FIFO; Otherwise, does not flush (false) the Rx FIFO

11.2.7.22 static void DSPI_SetAllPcsPolarity (SPI_Type * *base*, uint32_t *mask*) [inline], [static]

For example, PCS0 and PCS1 are set to active low and other PCS is set to active high. Note that the number of PCSs is specific to the device.

```
* DSPI_SetAllPcsPolarity(base, kDSPI_Pcs0ActiveLow |
    kDSPI_Pcs1ActiveLow);
```

Parameters

base	DSPI peripheral address.
mask	The PCS polarity mask; use the enum _dspi_pcs_polarity.

11.2.7.23 uint32_t DSPI_MasterSetBaudRate (SPI_Type ∗ *base,* dspi_ctar_selection_t *whichCtar,* uint32_t *baudRate_Bps,* uint32_t *srcClock_Hz*)

This function takes in the desired baudRate_Bps (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 that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

base	DSPI peripheral address.
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of the type dspi_ctar_-selection_t
baudRate_Bps	The desired baud rate in bits per second
srcClock_Hz	Module source input clock in Hertz

Returns

The actual calculated baud rate

11.2.7.24 void DSPI_MasterSetDelayScaler (SPI_Type ∗ *base,* dspi_ctar_selection_t *whichCtar,* uint32_t *prescaler,* uint32_t *scaler,* dspi_delay_type_t *whichDelay*)

This function configures the PCS to SCK delay pre-scalar (PcsSCK) and scalar (CSSCK), after SCK delay pre-scalar (PASC) and scalar (ASC), and the delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type dspi_delay_type_t.

The user passes the delay to the configuration along with the prescaler and scaler value. This allows the user to directly set the prescaler/scaler values if pre-calculated or to manually increment either value.

Parameters

base	DSPI peripheral address.
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_-selection_t.

prescaler	The prescaler delay value (can be an integer 0, 1, 2, or 3).
scaler	The scaler delay value (can be any integer between 0 to 15).
whichDelay	The desired delay to configure; must be of type dspi_delay_type_t

11.2.7.25 uint32_t DSPI_MasterSetDelayTimes (SPI_Type * *base,* dspi_ctar_selection_t *whichCtar,* dspi_delay_type_t *whichDelay,* uint32_t *srcClock_Hz,* uint32_t *delayTimeInNanoSec*)

This function calculates the values for the following. PCS to SCK delay pre-scalar (PCSSCK) and scalar (CSSCK), or After SCK delay pre-scalar (PASC) and scalar (ASC), or Delay after transfer pre-scalar (PDT) and scalar (DT).

These delay names are available in the type dspi_delay_type_t.

The user passes which delay to configure along with the desired delay value in nanoseconds. The function calculates the values needed for the prescaler and scaler. Note that returning the calculated delay as 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. The higher-level peripheral driver alerts the user of an out of range delay input.

Parameters

base	DSPI peripheral address.
whichCtar	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_selection_t.
whichDelay	The desired delay to configure, must be of type dspi_delay_type_t
srcClock_Hz	Module source input clock in Hertz
delayTimeInNanoSec	The desired delay value in nanoseconds.

Returns

The actual calculated delay value.

11.2.7.26 static void DSPI_MasterWriteData (SPI_Type * *base,* dspi_command_data_config_t * *command,* uint16_t *data*) [inline], [static]

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the

desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
* dspi_command_data_config_t commandConfig;
* commandConfig.isPcsContinuous = true;
* commandConfig.whichCtar = kDSPICtar0;
* commandConfig.whichPcs = kDSPIPcs0;
* commandConfig.clearTransferCount = false;
* commandConfig.isEndOfQueue = false;
* DSPI_MasterWriteData(base, &commandConfig, dataWord);
```

Parameters

base	DSPI peripheral address.
command	Pointer to the command structure.
data	The data word to be sent.

11.2.7.27 void DSPI_GetDefaultDataCommandConfig (dspi_command_data_config_t * *command*)

The purpose of this API is to get the configuration structure initialized for use in the DSPI_MasterWrite_xx(). Users may use the initialized structure unchanged in the DSPI_MasterWrite_xx() or modify the structure before calling the DSPI_MasterWrite_xx(). This is an example.

```
* dspi_command_data_config_t  command;
* DSPI_GetDefaultDataCommandConfig(&command);
*
```

Parameters

command	Pointer to the dspi_command_data_config_t structure.

11.2.7.28 void DSPI_MasterWriteDataBlocking (SPI_Type * *base,* dspi_command_data_config_t * *command,* uint16_t *data*)

In master mode, the 16-bit data is appended to the 16-bit command info. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). This is an example.

```
* dspi_command_config_t commandConfig;
* commandConfig.isPcsContinuous = true;
* commandConfig.whichCtar = kDSPICtar0;
```

```
*  commandConfig.whichPcs = kDSPIPcs1;
*  commandConfig.clearTransferCount = false;
*  commandConfig.isEndOfQueue = false;
*  DSPI_MasterWriteDataBlocking(base, &commandConfig, dataWord);
*
```

Note that this function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

Parameters

base	DSPI peripheral address.
command	Pointer to the command structure.
data	The data word to be sent.

11.2.7.29 static uint32_t DSPI_MasterGetFormattedCommand (dspi_command_data_-config_t * *command*) [inline], [static]

This function allows the caller to pass in the data command structure and returns the command word formatted according to the DSPI PUSHR register bit field placement. The user can then "OR" the returned command word with the desired data to send and use the function DSPI_HAL_WriteCommandDataMastermode or DSPI_HAL_WriteCommandDataMastermodeBlocking to write the entire 32-bit command data word to the PUSHR. This helps improve performance in cases where the command structure is constant. For example, the user calls this function before starting a transfer to generate the command word. When they are ready to transmit the data, they OR this formatted command word with the desired data to transmit. This process increases transmit performance when compared to calling send functions, such as DSPI_HAL_WriteDataMastermode, which format the command word each time a data word is to be sent.

Parameters

command	Pointer to the command structure.

Returns

The command word formatted to the PUSHR data register bit field.

11.2.7.30 void DSPI_MasterWriteCommandDataBlocking (SPI_Type * *base,* uint32_t *data*)

In this function, the user must append the 16-bit data to the 16-bit command information and then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data, such as the optional continuous chip select operation between transfers, the desired Clock and Transfer Attributes

register to use for the associated SPI frame, the desired PCS signal to use for the data transfer, whether the current transfer is the last in the queue, and whether to clear the transfer count (normally needed when sending the first frame of a data packet). The user is responsible for appending this command with the data to send. This is an example:

```
*  dataWord = <16-bit command> | <16-bit data>;
*  DSPI_MasterWriteCommandDataBlocking(base, dataWord);
*
```

Note that this function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running to transmit data (MCR[MDIS] & [HALT] = 0). Because the SPI is a synchronous protocol, the received data is available when the transmit completes.

For a blocking polling transfer, see methods below. Option 1: uint32_t command_to_send = DSPI_-MasterGetFormattedCommand(&command); uint32_t data0 = command_to_send | data_need_to_send-_0; uint32_t data1 = command_to_send | data_need_to_send_1; uint32_t data2 = command_to_send | data_need_to_send_2;

DSPI_MasterWriteCommandDataBlocking(base,data0); DSPI_MasterWriteCommandDataBlocking(base,data1); DSPI_MasterWriteCommandDataBlocking(base,data2);

Option 2: DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_0); DSPI_Master-WriteDataBlocking(base,&command,data_need_to_send_1); DSPI_MasterWriteDataBlocking(base,&command,data_need_to_send_2);

Parameters

base	DSPI peripheral address.
data	The data word (command and data combined) to be sent.

11.2.7.31 static void DSPI_SlaveWriteData (SPI_Type * *base,* uint32_t *data*) [inline], [static]

In slave mode, up to 16-bit words may be written.

Parameters

base	DSPI peripheral address.
data	The data to send.

11.2.7.32 void DSPI_SlaveWriteDataBlocking (SPI_Type * *base,* uint32_t *data*)

In slave mode, up to 16-bit words may be written. The function first clears the transmit complete flag, writes data into data register, and finally waits until the data is transmitted.

Parameters

base	DSPI peripheral address.
data	The data to send.

11.2.7.33 static uint32_t DSPI_ReadData (SPI_Type * *base*) [inline], [static]

Parameters

base	DSPI peripheral address.

Returns

The data from the read data buffer.

11.2.7.34 void DSPI_MasterTransferCreateHandle (SPI_Type * *base,* dspi_master_-handle_t * *handle,* dspi_master_transfer_callback_t *callback,* void * *userData*)

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

Parameters

base	DSPI peripheral base address.
handle	DSPI handle pointer to dspi_master_handle_t.
callback	DSPI callback.
userData	Callback function parameter.

11.2.7.35 status_t DSPI_MasterTransferBlocking (SPI_Type * *base,* dspi_transfer_t * *transfer*)

This function transfers data using polling. This is a blocking function, which does not return until all transfers have been completed.

Parameters

base	DSPI peripheral base address.
transfer	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.2.7.36 status_t DSPI_MasterTransferNonBlocking (SPI_Type * *base*, dspi_master_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using interrupts. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_master_handle_t structure which stores the transfer state.
transfer	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.2.7.37 status_t DSPI_MasterTransferGetCount (SPI_Type * *base*, dspi_master_handle_t * *handle*, size_t * *count*)

This function gets the master transfer count.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_master_handle_t structure which stores the transfer state.
count	The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

11.2.7.38 void DSPI_MasterTransferAbort (SPI_Type * *base,* dspi_master_handle_t * *handle*)

This function aborts a transfer using an interrupt.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_master_handle_t structure which stores the transfer state.

11.2.7.39 void DSPI_MasterTransferHandleIRQ (SPI_Type * *base,* dspi_master_handle_t * *handle*)

This function processes the DSPI transmit and receive IRQ.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_master_handle_t structure which stores the transfer state.

11.2.7.40 void DSPI_SlaveTransferCreateHandle (SPI_Type * *base,* dspi_slave_handle_t * *handle,* dspi_slave_transfer_callback_t *callback,* void * *userData*)

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

Parameters

handle	DSPI handle pointer to the dspi_slave_handle_t.
base	DSPI peripheral base address.
callback	DSPI callback.
userData	Callback function parameter.

11.2.7.41 status_t DSPI_SlaveTransferNonBlocking (SPI_Type * *base,* dspi_slave_handle_t * *handle,* dspi_transfer_t * *transfer*)

This function transfers data using an interrupt. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_slave_handle_t structure which stores the transfer state.
transfer	Pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.2.7.42 status_t DSPI_SlaveTransferGetCount (SPI_Type * *base*, dspi_slave_handle_t * *handle*, size_t * *count*)

This function gets the slave transfer count.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_master_handle_t structure which stores the transfer state.
count	The number of bytes transferred by using the non-blocking transaction.

Returns

status of status_t.

11.2.7.43 void DSPI_SlaveTransferAbort (SPI_Type * *base*, dspi_slave_handle_t * *handle*)

This function aborts a transfer using an interrupt.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_slave_handle_t structure which stores the transfer state.

11.2.7.44 void DSPI_SlaveTransferHandleIRQ (SPI_Type * *base*, dspi_slave_handle_t * *handle*)

This function processes the DSPI transmit and receive IRQ.

Parameters

base	DSPI peripheral base address.
handle	Pointer to the dspi_slave_handle_t structure which stores the transfer state.

11.3 DSPI DMA Driver

11.3.1 Overview

This section describes the programming interface of the DSPI Peripheral driver. The DSPI driver configures DSPI module and provides the functional and transactional interfaces to build the DSPI application.

Data Structures

- struct dspi_master_dma_handle_t
  DSPI master DMA transfer handle structure used for transactional API. More...
- struct dspi_slave_dma_handle_t
  DSPI slave DMA transfer handle structure used for transactional API. More...

Typedefs

- typedef void(∗ dspi_master_dma_transfer_callback_t)(SPI_Type ∗base, dspi_master_dma_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.
- typedef void(∗ dspi_slave_dma_transfer_callback_t)(SPI_Type ∗base, dspi_slave_dma_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.

Functions

- void DSPI_MasterTransferCreateHandleDMA (SPI_Type ∗base, dspi_master_dma_handle_t ∗handle, dspi_master_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗dmaRxRegToRxDataHandle, dma_handle_t ∗dmaTxDataToIntermediaryHandle, dma_handle_t ∗dmaIntermediaryToTxRegHandle)
  Initializes the DSPI master DMA handle.
- status_t DSPI_MasterTransferDMA (SPI_Type ∗base, dspi_master_dma_handle_t ∗handle, dspi_transfer_t ∗transfer)
  DSPI master transfers data using DMA.
- void DSPI_MasterTransferAbortDMA (SPI_Type ∗base, dspi_master_dma_handle_t ∗handle)
  DSPI master aborts a transfer which is using DMA.
- status_t DSPI_MasterTransferGetCountDMA (SPI_Type ∗base, dspi_master_dma_handle_t ∗handle, size_t ∗count)
  Gets the master DMA transfer remaining bytes.
- void DSPI_SlaveTransferCreateHandleDMA (SPI_Type ∗base, dspi_slave_dma_handle_t ∗handle, dspi_slave_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗dmaRxRegToRxDataHandle, dma_handle_t ∗dmaTxDataToTxRegHandle)
  Initializes the DSPI slave DMA handle.
- status_t DSPI_SlaveTransferDMA (SPI_Type ∗base, dspi_slave_dma_handle_t ∗handle, dspi_transfer_t ∗transfer)
  DSPI slave transfers data using DMA.

- void DSPI_SlaveTransferAbortDMA (SPI_Type *base, dspi_slave_dma_handle_t *handle)
  DSPI slave aborts a transfer which is using DMA.
- status_t DSPI_SlaveTransferGetCountDMA (SPI_Type *base, dspi_slave_dma_handle_t *handle, size_t *count)
  Gets the slave DMA transfer remaining bytes.

11.3.2 Data Structure Documentation

11.3.2.1 struct _dspi_master_dma_handle

Forward declaration of the DSPI DMA master handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile uint32_t command
  The desired data command.
- volatile uint32_t lastCommand
  The desired last data command.
- uint8_t fifoSize
  FIFO dataSize.
- volatile bool isPcsActiveAfterTransfer
  Indicates whether the PCS signal keeps active after the last frame transfer.
- volatile bool isThereExtraByte
  Indicates whether there is an extra byte.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.
- uint32_t rxBuffIfNull
  Used if there is not rxData for DMA purpose.
- uint32_t txBuffIfNull
  Used if there is not txData for DMA purpose.
- volatile uint8_t state
  DSPI transfer state, see _dspi_transfer_state.
- dspi_master_dma_transfer_callback_t callback
  Completion callback.
- void * userData
  Callback user data.
- dma_handle_t * dmaRxRegToRxDataHandle
  dma_handle_t handle point used for RxReg to RxData buff

- dma_handle_t * dmaTxDataToIntermediaryHandle
  dma_handle_t handle point used for TxData to Intermediary
- dma_handle_t * dmaIntermediaryToTxRegHandle
  dma_handle_t handle point used for Intermediary to TxReg

11.3.2.1.0.21 Field Documentation

11.3.2.1.0.21.1 uint32_t dspi_master_dma_handle_t::bitsPerFrame

11.3.2.1.0.21.2 volatile uint32_t dspi_master_dma_handle_t::command

11.3.2.1.0.21.3 volatile uint32_t dspi_master_dma_handle_t::lastCommand

11.3.2.1.0.21.4 uint8_t dspi_master_dma_handle_t::fifoSize

11.3.2.1.0.21.5 volatile bool dspi_master_dma_handle_t::isPcsActiveAfterTransfer

11.3.2.1.0.21.6 volatile bool dspi_master_dma_handle_t::isThereExtraByte

11.3.2.1.0.21.7 uint8_t* volatile dspi_master_dma_handle_t::txData

11.3.2.1.0.21.8 uint8_t* volatile dspi_master_dma_handle_t::rxData

11.3.2.1.0.21.9 volatile size_t dspi_master_dma_handle_t::remainingSendByteCount

11.3.2.1.0.21.10 volatile size_t dspi_master_dma_handle_t::remainingReceiveByteCount

11.3.2.1.0.21.11 uint32_t dspi_master_dma_handle_t::rxBuffIfNull

11.3.2.1.0.21.12 uint32_t dspi_master_dma_handle_t::txBuffIfNull

11.3.2.1.0.21.13 volatile uint8_t dspi_master_dma_handle_t::state

11.3.2.1.0.21.14 dspi_master_dma_transfer_callback_t dspi_master_dma_handle_t::callback

11.3.2.1.0.21.15 void* dspi_master_dma_handle_t::userData

11.3.2.2 struct _dspi_slave_dma_handle

Forward declaration of the DSPI DMA slave handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  Desired number of bits per frame.
- volatile bool isThereExtraByte
  Indicates whether there is an extra byte.
- uint8_t *volatile txData
  A send buffer.
- uint8_t *volatile rxData

A receive buffer.

- volatile size_t remainingSendByteCount

  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount

  A number of bytes remaining to receive.
- size_t totalByteCount

  A number of transfer bytes.
- uint32_t rxBuffIfNull

  Used if there is not rxData for DMA purpose.
- uint32_t txBuffIfNull

  Used if there is not txData for DMA purpose.
- uint32_t txLastData

  Used if there is an extra byte when 16 bits per frame for DMA purpose.
- volatile uint8_t state

  DSPI transfer state.
- uint32_t errorCount

  Error count for the slave transfer.
- dspi_slave_dma_transfer_callback_t callback

  Completion callback.
- void * userData

  Callback user data.
- dma_handle_t * dmaRxRegToRxDataHandle

  dma_handle_t handle point used for RxReg to RxData buff
- dma_handle_t * dmaTxDataToTxRegHandle

  dma_handle_t handle point used for TxData to TxReg

11.3.2.2.0.22 Field Documentation

11.3.2.2.0.22.1 uint32_t dspi_slave_dma_handle_t::bitsPerFrame

11.3.2.2.0.22.2 volatile bool dspi_slave_dma_handle_t::isThereExtraByte

11.3.2.2.0.22.3 uint8_t∗ volatile dspi_slave_dma_handle_t::txData

11.3.2.2.0.22.4 uint8_t∗ volatile dspi_slave_dma_handle_t::rxData

11.3.2.2.0.22.5 volatile size_t dspi_slave_dma_handle_t::remainingSendByteCount

11.3.2.2.0.22.6 volatile size_t dspi_slave_dma_handle_t::remainingReceiveByteCount

11.3.2.2.0.22.7 uint32_t dspi_slave_dma_handle_t::rxBuffIfNull

11.3.2.2.0.22.8 uint32_t dspi_slave_dma_handle_t::txBuffIfNull

11.3.2.2.0.22.9 uint32_t dspi_slave_dma_handle_t::txLastData

11.3.2.2.0.22.10 volatile uint8_t dspi_slave_dma_handle_t::state

11.3.2.2.0.22.11 uint32_t dspi_slave_dma_handle_t::errorCount

11.3.2.2.0.22.12 dspi_slave_dma_transfer_callback_t dspi_slave_dma_handle_t::callback

11.3.2.2.0.22.13 void∗ dspi_slave_dma_handle_t::userData

11.3.3 Typedef Documentation

11.3.3.1 typedef void(∗ dspi_master_dma_transfer_callback_t)(SPI_Type ∗base, dspi_master_dma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral base address.
handle	Pointer to the handle for the DSPI master.
status	Success or error code describing whether the transfer completed.
userData	Arbitrary pointer-dataSized value passed from the application.

11.3.3.2 typedef void(∗ dspi_slave_dma_transfer_callback_t)(SPI_Type ∗base, dspi_slave_dma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral base address.
handle	Pointer to the handle for the DSPI slave.
status	Success or error code describing whether the transfer completed.
userData	Arbitrary pointer-dataSized value passed from the application.

11.3.4 Function Documentation

11.3.4.1 void DSPI_MasterTransferCreateHandleDMA (SPI_Type ∗ *base*, dspi_master_dma_handle_t ∗ *handle*, dspi_master_dma_transfer_callback_t *callback*, void ∗ *userData*, dma_handle_t ∗ *dmaRxRegToRxDataHandle*, dma_handle_t ∗ *dmaTxDataToIntermediaryHandle*, dma_handle_t ∗ *dmaIntermediaryToTxRegHandle*)

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

Note that DSPI DMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx is the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for dmaRxRegToRxDataHandle and Tx DMAMUX source for dmaIntermediaryToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for dmaRxRegToRxDataHandle.

Parameters

base	DSPI peripheral base address.
handle	DSPI handle pointer to dspi_master_dma_handle_t.
callback	DSPI callback.
userData	A callback function parameter.
dmaRxRegToRxDataHandle	dmaRxRegToRxDataHandle pointer to dma_handle_t.
dmaTxDataToIntermediaryHandle	dmaTxDataToIntermediaryHandle pointer to dma_handle_t.
dmaIntermediaryToTxRegHandle	dmaIntermediaryToTxRegHandle pointer to dma_handle_t.

11.3.4.2 status_t DSPI_MasterTransferDMA (SPI_Type * *base*, dspi_master_dma_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using DMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note that the master DMA transfer does not support the transfer_size of 1 when the bitsPerFrame is greater than 8.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_master_dma_handle_t structure which stores the transfer state.
transfer	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.3.4.3 void DSPI_MasterTransferAbortDMA (SPI_Type * *base*, dspi_master_dma_handle_t * *handle*)

This function aborts a transfer which is using DMA.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_master_dma_handle_t structure which stores the transfer state.

11.3.4.4 status_t DSPI_MasterTransferGetCountDMA (SPI_Type * *base*, dspi_master_dma_handle_t * *handle*, size_t * *count*)

This function gets the master DMA transfer remaining bytes.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_master_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

11.3.4.5 void DSPI_SlaveTransferCreateHandleDMA (SPI_Type * *base*, dspi_slave_dma_handle_t * *handle*, dspi_slave_dma_transfer_callback_t *callback*, void * *userData*, dma_handle_t * *dmaRxRegToRxDataHandle*, dma_handle_t * *dmaTxDataToTxRegHandle*)

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

Note that DSPI DMA has a separated (Rx and Tx as two sources) or shared (Rx and Tx is the same source) DMA request source. (1) For a separated DMA request source, enable and set the Rx DMAMUX source for dmaRxRegToRxDataHandle and Tx DMAMUX source for dmaTxDataToTxRegHandle. (2) For a shared DMA request source, enable and set the Rx/Rx DMAMUX source for dmaRxRegToRxDataHandle.

Parameters

base	DSPI peripheral base address.

handle	DSPI handle pointer to dspi_slave_dma_handle_t.
callback	DSPI callback.
userData	A callback function parameter.
dmaRxRegToRxDataHandle	dmaRxRegToRxDataHandle pointer to dma_handle_t.
dmaTxDataToTxRegHandle	dmaTxDataToTxRegHandle pointer to dma_handle_t.

11.3.4.6 status_t DSPI_SlaveTransferDMA (SPI_Type * *base*, dspi_slave_dma_handle_t * *handle*, dspi_transfer_t * *transfer*)

This function transfers data using DMA. This is a non-blocking function, which returns right away. When all data is transferred, the callback function is called.

Note that the slave DMA transfer does not support the transfer_size of 1 when the bitsPerFrame is greater than eight.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_slave_dma_handle_t structure which stores the transfer state.
transfer	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.3.4.7 void DSPI_SlaveTransferAbortDMA (SPI_Type * *base*, dspi_slave_dma_handle_t * *handle*)

This function aborts a transfer which is using DMA.

Parameters

base	DSPI peripheral base address.

handle	A pointer to the dspi_slave_dma_handle_t structure which stores the transfer state.

11.3.4.8 status_t DSPI_SlaveTransferGetCountDMA (SPI_Type * *base,* dspi_slave_dma_handle_t * *handle,* size_t * *count*)

This function gets the slave DMA transfer remaining bytes.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_slave_dma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

11.4 DSPI eDMA Driver

11.4.1 Overview

This section describes the programming interface of the DSPI Peripheral driver. The DSPI driver configures DSPI module and provides the functional and transactional interfaces to build the DSPI application.

Data Structures

- struct dspi_master_edma_handle_t
  DSPI master eDMA transfer handle structure used for the transactional API. More...
- struct dspi_slave_edma_handle_t
  DSPI slave eDMA transfer handle structure used for the transactional API. More...

Typedefs

- typedef void(∗ dspi_master_edma_transfer_callback_t)(SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.
- typedef void(∗ dspi_slave_edma_transfer_callback_t)(SPI_Type ∗base, dspi_slave_edma_handle_t ∗handle, status_t status, void ∗userData)
  Completion callback function pointer type.

Functions

- void DSPI_MasterTransferCreateHandleEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, dspi_master_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edmaRxRegToRxDataHandle, edma_handle_t ∗edmaTxDataToIntermediaryHandle, edma_handle_t ∗edmaIntermediaryToTxRegHandle)
  Initializes the DSPI master eDMA handle.
- status_t DSPI_MasterTransferEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, dspi_transfer_t ∗transfer)
  DSPI master transfer data using eDMA.
- void DSPI_MasterTransferAbortEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle)
  DSPI master aborts a transfer which is using eDMA.
- status_t DSPI_MasterTransferGetCountEDMA (SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, size_t ∗count)
  Gets the master eDMA transfer count.
- void DSPI_SlaveTransferCreateHandleEDMA (SPI_Type ∗base, dspi_slave_edma_handle_t ∗handle, dspi_slave_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edmaRxRegToRxDataHandle, edma_handle_t ∗edmaTxDataToTxRegHandle)
  Initializes the DSPI slave eDMA handle.
- status_t DSPI_SlaveTransferEDMA (SPI_Type ∗base, dspi_slave_edma_handle_t ∗handle, dspi_transfer_t ∗transfer)
  DSPI slave transfer data using eDMA.

- void DSPI_SlaveTransferAbortEDMA (SPI_Type *base, dspi_slave_edma_handle_t *handle)
  DSPI slave aborts a transfer which is using eDMA.
- status_t DSPI_SlaveTransferGetCountEDMA (SPI_Type *base, dspi_slave_edma_handle_t *handle, size_t *count)
  Gets the slave eDMA transfer count.

11.4.2 Data Structure Documentation

11.4.2.1 struct _dspi_master_edma_handle

Forward declaration of the DSPI eDMA master handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- volatile uint32_t command
  The desired data command.
- volatile uint32_t lastCommand
  The desired last data command.
- uint8_t fifoSize
  FIFO dataSize.
- volatile bool isPcsActiveAfterTransfer
  Indicates whether the PCS signal keeps active after the last frame transfer.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- volatile uint8_t state
  DSPI transfer state , _dspi_transfer_state.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData
  Receive buffer.
- volatile size_t remainingSendByteCount
  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount
  A number of bytes remaining to receive.
- size_t totalByteCount
  A number of transfer bytes.
- uint32_t rxBuffIfNull
  Used if there is not rxData for DMA purpose.
- uint32_t txBuffIfNull
  Used if there is not txData for DMA purpose.
- dspi_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 * edmaTxDataToIntermediaryHandle
  edma_handle_t handle point used for TxData to Intermediary
- edma_handle_t * edmaIntermediaryToTxRegHandle
  edma_handle_t handle point used for Intermediary to TxReg
- edma_tcd_t dspiSoftwareTCD [2]
  SoftwareTCD , internal used.

11.4.2.1.0.23 Field Documentation

11.4.2.1.0.23.1 uint32_t dspi_master_edma_handle_t::bitsPerFrame

11.4.2.1.0.23.2 volatile uint32_t dspi_master_edma_handle_t::command

11.4.2.1.0.23.3 volatile uint32_t dspi_master_edma_handle_t::lastCommand

11.4.2.1.0.23.4 uint8_t dspi_master_edma_handle_t::fifoSize

11.4.2.1.0.23.5 volatile bool dspi_master_edma_handle_t::isPcsActiveAfterTransfer

11.4.2.1.0.23.6 uint8_t dspi_master_edma_handle_t::nbytes

11.4.2.1.0.23.7 volatile uint8_t dspi_master_edma_handle_t::state

11.4.2.1.0.23.8 uint8_t* volatile dspi_master_edma_handle_t::txData

11.4.2.1.0.23.9 uint8_t* volatile dspi_master_edma_handle_t::rxData

11.4.2.1.0.23.10 volatile size_t dspi_master_edma_handle_t::remainingSendByteCount

11.4.2.1.0.23.11 volatile size_t dspi_master_edma_handle_t::remainingReceiveByteCount

11.4.2.1.0.23.12 uint32_t dspi_master_edma_handle_t::rxBuffIfNull

11.4.2.1.0.23.13 uint32_t dspi_master_edma_handle_t::txBuffIfNull

11.4.2.1.0.23.14 dspi_master_edma_transfer_callback_t dspi_master_edma_handle_t::callback

11.4.2.1.0.23.15 void* dspi_master_edma_handle_t::userData

11.4.2.2 struct _dspi_slave_edma_handle

Forward declaration of the DSPI eDMA slave handle typedefs.

Data Fields

- uint32_t bitsPerFrame
  The desired number of bits per frame.
- uint8_t *volatile txData
  Send buffer.
- uint8_t *volatile rxData

Receive buffer.

- volatile size_t remainingSendByteCount

  A number of bytes remaining to send.
- volatile size_t remainingReceiveByteCount

  A number of bytes remaining to receive.
- size_t totalByteCount

  A number of transfer bytes.
- uint32_t rxBuffIfNull

  Used if there is not rxData for DMA purpose.
- uint32_t txBuffIfNull

  Used if there is not txData for DMA purpose.
- uint32_t txLastData

  Used if there is an extra byte when 16bits per frame for DMA purpose.
- uint8_t nbytes

  eDMA minor byte transfer count initially configured.
- volatile uint8_t state

  DSPI transfer state.
- dspi_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

11.4.2.2.0.24 Field Documentation

11.4.2.2.0.24.1 uint32_t dspi_slave_edma_handle_t::bitsPerFrame

11.4.2.2.0.24.2 uint8_t∗ volatile dspi_slave_edma_handle_t::txData

11.4.2.2.0.24.3 uint8_t∗ volatile dspi_slave_edma_handle_t::rxData

11.4.2.2.0.24.4 volatile size_t dspi_slave_edma_handle_t::remainingSendByteCount

11.4.2.2.0.24.5 volatile size_t dspi_slave_edma_handle_t::remainingReceiveByteCount

11.4.2.2.0.24.6 uint32_t dspi_slave_edma_handle_t::rxBuffIfNull

11.4.2.2.0.24.7 uint32_t dspi_slave_edma_handle_t::txBuffIfNull

11.4.2.2.0.24.8 uint32_t dspi_slave_edma_handle_t::txLastData

11.4.2.2.0.24.9 uint8_t dspi_slave_edma_handle_t::nbytes

11.4.2.2.0.24.10 volatile uint8_t dspi_slave_edma_handle_t::state

11.4.2.2.0.24.11 dspi_slave_edma_transfer_callback_t dspi_slave_edma_handle_t::callback

11.4.2.2.0.24.12 void∗ dspi_slave_edma_handle_t::userData

11.4.3 Typedef Documentation

11.4.3.1 typedef void(∗ dspi_master_edma_transfer_callback_t)(SPI_Type ∗base, dspi_master_edma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral base address.
handle	A pointer to the handle for the DSPI master.
status	Success or error code describing whether the transfer completed.
userData	An arbitrary pointer-dataSized value passed from the application.

11.4.3.2 typedef void(∗ dspi_slave_edma_transfer_callback_t)(SPI_Type ∗base, dspi_slave_edma_handle_t ∗handle, status_t status, void ∗userData)

Parameters

base	DSPI peripheral base address.
handle	A pointer to the handle for the DSPI slave.
status	Success or error code describing whether the transfer completed.
userData	An arbitrary pointer-dataSized value passed from the application.

11.4.4 Function Documentation

11.4.4.1 void DSPI_MasterTransferCreateHandleEDMA (SPI_Type ∗ *base*, dspi_master_edma_handle_t ∗ *handle*, dspi_master_edma_transfer_callback_t *callback*, void ∗ *userData*, edma_handle_t ∗ *edmaRxRegToRxDataHandle*, edma_handle_t ∗ *edmaTxDataToIntermediaryHandle*, edma_handle_t ∗ *edmaIntermediaryToTxRegHandle*)

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

Note that DSPI eDMA has separated (RX and TX as two sources) or shared (RX and TX are the same source) DMA request source. (1) For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaIntermediaryToTxRegHandle. (2) For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

base	DSPI peripheral base address.
handle	DSPI handle pointer to dspi_master_edma_handle_t.
callback	DSPI callback.
userData	A callback function parameter.
edmaRxRegToRxDataHandle	edmaRxRegToRxDataHandle pointer to edma_handle_t.
edmaTxDataToIntermediaryHandle	edmaTxDataToIntermediaryHandle pointer to edma_handle_t.
edmaIntermediaryToTxRegHandle	edmaIntermediaryToTxRegHandle pointer to edma_handle_t.

11.4.4.2 status_t DSPI_MasterTransferEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*, dspi_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.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_master_edma_handle_t structure which stores the transfer state.
transfer	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.4.4.3 void DSPI_MasterTransferAbortEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*)

This function aborts a transfer which is using eDMA.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_master_edma_handle_t structure which stores the transfer state.

11.4.4.4 status_t DSPI_MasterTransferGetCountEDMA (SPI_Type * *base*, dspi_master_edma_handle_t * *handle*, size_t * *count*)

This function gets the master eDMA transfer count.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_master_edma_handle_t structure which stores the transfer state.
count	A number of bytes transferred by the non-blocking transaction.

Returns

status of status_t.

11.4.4.5 void DSPI_SlaveTransferCreateHandleEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, dspi_slave_edma_transfer_callback_t *callback*, void * *userData*, edma_handle_t * *edmaRxRegToRxDataHandle*, edma_handle_t * *edmaTxDataToTxRegHandle*)

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

Note that DSPI eDMA has separated (RN and TX in 2 sources) or shared (RX and TX are the same source) DMA request source. (1)For the separated DMA request source, enable and set the RX DMAMUX source for edmaRxRegToRxDataHandle and TX DMAMUX source for edmaTxDataToTxRegHandle. (2)For the shared DMA request source, enable and set the RX/RX DMAMUX source for the edmaRxRegToRxDataHandle.

Parameters

base	DSPI peripheral base address.

handle	DSPI handle pointer to dspi_slave_edma_handle_t.
callback	DSPI callback.
userData	A callback function parameter.
edmaRxRegToRxDataHandle	edmaRxRegToRxDataHandle pointer to edma_handle_t.
edmaTxDataToTxRegHandle	edmaTxDataToTxRegHandle pointer to edma_handle_t.

11.4.4.6 status_t DSPI_SlaveTransferEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, dspi_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 that the slave eDMA transfer doesn't support transfer_size is 1 when the bitsPerFrame is greater than eight.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_slave_edma_handle_t structure which stores the transfer state.
transfer	A pointer to the dspi_transfer_t structure.

Returns

status of status_t.

11.4.4.7 void DSPI_SlaveTransferAbortEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*)

This function aborts a transfer which is using eDMA.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_slave_edma_handle_t structure which stores the transfer state.

11.4.4.8 status_t DSPI_SlaveTransferGetCountEDMA (SPI_Type * *base*, dspi_slave_edma_handle_t * *handle*, size_t * *count*)

This function gets the slave eDMA transfer count.

Parameters

base	DSPI peripheral base address.
handle	A pointer to the dspi_slave_edma_handle_t structure which stores the transfer state.
count	A number of bytes transferred so far by the non-blocking transaction.

Returns

status of status_t.

11.5 DSPI FreeRTOS Driver

11.5.1 Overview

DSPI RTOS Operation

- status_t DSPI_RTOS_Init (dspi_rtos_handle_t *handle, SPI_Type *base, const dspi_master_config_t *masterConfig, uint32_t srcClock_Hz)
  Initializes the DSPI.
- status_t DSPI_RTOS_Deinit (dspi_rtos_handle_t *handle)
  Deinitializes the DSPI.
- status_t DSPI_RTOS_Transfer (dspi_rtos_handle_t *handle, dspi_transfer_t *transfer)
  Performs the SPI transfer.

11.5.2 Function Documentation

11.5.2.1 status_t DSPI_RTOS_Init (dspi_rtos_handle_t * *handle,* SPI_Type * *base,* const dspi_master_config_t * *masterConfig,* uint32_t *srcClock_Hz*)

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

Parameters

handle	The RTOS DSPI handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the DSPI instance to initialize.
masterConfig	A configuration structure to set-up the DSPI in master mode.
srcClock_Hz	A frequency of the input clock of the DSPI module.

Returns

status of the operation.

11.5.2.2 status_t DSPI_RTOS_Deinit (dspi_rtos_handle_t * *handle*)

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

Parameters

handle	The RTOS DSPI handle.

11.5.2.3 status_t DSPI_RTOS_Transfer (dspi_rtos_handle_t * *handle,* dspi_transfer_t * *transfer*)

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

Parameters

handle	The RTOS DSPI handle.
transfer	A structure specifying the transfer parameters.

Returns

status of the operation.

Chapter 12
eDMA: Enhanced Direct Memory Access (eDMA) Controller Driver

12.1 Overview

The MCUXpresso SDK provides a peripheral driver for the enhanced Direct Memory Access (eDMA) of MCUXpresso SDK devices.

12.2 Typical use case

12.2.1 eDMA Operation

```
edma_transfer_config_t transferConfig;
edma_config_t userConfig;
uint32_t transferDone = false;

EDMA_GetDefaultConfig(&userConfig);
EDMA_Init(DMA0, &userConfig);
EDMA_CreateHandle(&g_EDMA_Handle, DMA0, channel);
EDMA_SetCallback(&g_EDMA_Handle, EDMA_Callback, &transferDone);
EDMA_PrepareTransfer(&transferConfig, srcAddr, srcWidth, destAddr, destWidth,
                     bytesEachRequest, transferBytes, kEDMA_MemoryToMemory);
EDMA_SubmitTransfer(&g_EDMA_Handle, &transferConfig, true);
EDMA_StartTransfer(&g_EDMA_Handle);
/* Waits for the eDMA transfer to finish */
while (transferDone != true);
```

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) | (3 - ((channel)&0x03U)))
  Compute the offset unit from DCHPRI3.
- #define DMA_DCHPRIn(base, channel) ((volatile uint8_t *)&(base->DCHPRI3))[DMA_DCHPRI_INDEX(channel)]
  Get the pointer of DCHPRIn.

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_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 _edma_channel_status_flags {

kEDMA_DoneFlag = 0x1U,
kEDMA_ErrorFlag = 0x2U,
kEDMA_InterruptFlag = 0x4U }
eDMA channel status flags.

- enum _edma_error_status_flags {
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 = DMA_ES_VLD_MASK }
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 }
eDMA transfer type
- enum _edma_transfer_status {
kStatus_EDMA_QueueFull = MAKE_STATUS(kStatusGroup_EDMA, 0),
kStatus_EDMA_Busy = MAKE_STATUS(kStatusGroup_EDMA, 1) }
eDMA transfer status

Driver version

- #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))
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.

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.
- static 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.

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.

Typical use case

- 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.

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_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.
- void EDMA_HandleIRQ (edma_handle_t *handle)
  eDMA IRQ handler for the current major loop transfer completion.

12.3 Data Structure Documentation

12.3.1 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.

12.3.1.0.0.25 Field Documentation

12.3.1.0.0.25.1 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.

12.3.1.0.0.25.2 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.

12.3.1.0.0.25.3 bool edma_config_t::enableDebugMode

When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete.

12.3.2 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.

12.3.2.0.0.26 Field Documentation

12.3.2.0.0.26.1 uint32_t edma_transfer_config_t::srcAddr

12.3.2.0.0.26.2 uint32_t edma_transfer_config_t::destAddr

12.3.2.0.0.26.3 edma_transfer_size_t edma_transfer_config_t::srcTransferSize

12.3.2.0.0.26.4 edma_transfer_size_t edma_transfer_config_t::destTransferSize

12.3.2.0.0.26.5 int16_t edma_transfer_config_t::srcOffset

12.3.2.0.0.26.6 int16_t edma_transfer_config_t::destOffset

12.3.2.0.0.26.7 uint32_t edma_transfer_config_t::majorLoopCounts

12.3.3 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.

12.3.4 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.

12.3.4.0.0.27 Field Documentation

12.3.4.0.0.27.1 bool edma_minor_offset_config_t::enableSrcMinorOffset

12.3.4.0.0.27.2 bool edma_minor_offset_config_t::enableDestMinorOffset

12.3.4.0.0.27.3 uint32_t edma_minor_offset_config_t::minorOffset

12.3.5 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
  ATTR register, source/destination transfer size and modulo.
- __IO uint32_t NBYTES
  Nbytes register, minor loop length in bytes.
- __IO uint32_t SLAST
  SLAST register.
- __IO uint32_t DADDR
  DADDR register, used for destination address.
- __IO uint16_t DOFF
  DOFF register, used for destination offset.
- __IO uint16_t CITER
  CITER register, current minor loop numbers, for unfinished minor loop.
- __IO uint32_t DLAST_SGA
  DLASTSGA register, next stcd address used in scatter-gather mode.
- __IO uint16_t CSR
  CSR register, for TCD control status.
- __IO uint16_t BITER
  BITER register, begin minor loop count.

12.3.5.0.0.28 Field Documentation

12.3.5.0.0.28.1 __IO uint16_t edma_tcd_t::CITER

12.3.5.0.0.28.2 __IO uint16_t edma_tcd_t::BITER

12.3.6 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.

12.3.6.0.0.29 Field Documentation

12.3.6.0.0.29.1 edma_callback edma_handle_t::callback

12.3.6.0.0.29.2 void* edma_handle_t::userData

12.3.6.0.0.29.3 DMA_Type* edma_handle_t::base

12.3.6.0.0.29.4 edma_tcd_t* edma_handle_t::tcdPool

12.3.6.0.0.29.5 uint8_t edma_handle_t::channel

12.3.6.0.0.29.6 volatile int8_t edma_handle_t::header

Should point to the next TCD to be loaded into the eDMA engine.

12.3.6.0.0.29.7 volatile int8_t edma_handle_t::tail

Should point to the next TCD to be stored into the memory pool.

12.3.6.0.0.29.8 volatile int8_t edma_handle_t::tcdUsed

Should reflect the number of TCDs can be used/loaded in the memory.

12.3.6.0.0.29.9 volatile int8_t edma_handle_t::tcdSize

12.3.6.0.0.29.10 uint8_t edma_handle_t::flags

12.4 Macro Definition Documentation

12.4.1 #define FSL_EDMA_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))

Version 2.1.1.

12.5 Typedef Documentation

12.5.1 typedef void(∗ edma_callback)(struct _edma_handle ∗handle, void ∗userData, bool transferDone, uint32_t tcds)

12.6 Enumeration Type Documentation

12.6.1 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_TransferSize16Bytes Source/Destination data transfer size is 16 bytes every time.
kEDMA_TransferSize32Bytes Source/Destination data transfer size is 32 bytes every time.

12.6.2 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.

12.6.3 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.

12.6.4 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.

12.6.5 enum _edma_channel_status_flags

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

12.6.6 enum _edma_error_status_flags

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.

12.6.7 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.

12.6.8 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.

12.6.9 enum _edma_transfer_status

Enumerator

kStatus_EDMA_QueueFull TCD queue is full.
kStatus_EDMA_Busy Channel is busy and can't handle the transfer request.

12.7 Function Documentation

12.7.1 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

base	eDMA peripheral base address.
config	A pointer to the configuration structure, see "edma_config_t".

Note

This function enables the minor loop map feature.

12.7.2 void EDMA_Deinit (DMA_Type * *base*)

This function gates the eDMA clock.

Parameters

base	eDMA peripheral base address.

12.7.3 void EDMA_InstallTCD (DMA_Type * *base,* uint32_t *channel,* edma_tcd_t * *tcd*)

Parameters

base	EDMA peripheral base address.
channel	EDMA channel number.
tcd	Point to TCD structure.

12.7.4 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

config	A pointer to the eDMA configuration structure.

12.7.5 void EDMA_ResetChannel (DMA_Type * *base,* uint32_t *channel*)

This function sets TCD registers for this channel to default values.

Parameters

base	eDMA peripheral base address.
channel	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.

12.7.6 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, &stcd);
*
```

Parameters

base	eDMA peripheral base address.

channel	eDMA channel number.
config	Pointer to eDMA transfer configuration structure.
nextTcd	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.

12.7.7 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

base	eDMA peripheral base address.
channel	eDMA channel number.
config	A pointer to the minor offset configuration structure.

12.7.8 static void EDMA_SetChannelPreemptionConfig (DMA_Type * *base,* uint32_t *channel,* const edma_channel_Preemption_config_t * *config*) `[inline], [static]`

This function configures the channel preemption attribute and the priority of the channel.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number
config	A pointer to the channel preemption configuration structure.

12.7.9 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

base	eDMA peripheral base address.
channel	eDMA channel number.
type	A channel link type, which can be one of the following: • kEDMA_LinkNone • kEDMA_MinorLink • kEDMA_MajorLink
linkedChannel	The linked channel number.

Note

Users should ensure that DONE flag is cleared before calling this interface, or the configuration is invalid.

12.7.10 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

base	eDMA peripheral base address.
channel	eDMA channel number.
bandWidth	A bandwidth setting, which can be one of the following: • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

12.7.11 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

base	eDMA peripheral base address.
channel	eDMA channel number.
srcModulo	A source modulo value.
destModulo	A destination modulo value.

12.7.12 static void EDMA_EnableAsyncRequest (DMA_Type * *base,* uint32_t *channel,* bool *enable*) [inline], [static]

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.
enable	The command to enable (true) or disable (false).

12.7.13 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

base	eDMA peripheral base address.
channel	eDMA channel number.
enable	The command to enable (true) or disable (false).

12.7.14 void EDMA_EnableChannelInterrupts (DMA_Type * *base,* uint32_t *channel,* uint32_t *mask*)

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.
mask	The mask of interrupt source to be set. Users need to use the defined edma_interrupt-_enable_t type.

12.7.15 void EDMA_DisableChannelInterrupts (DMA_Type * *base,* uint32_t *channel,* uint32_t *mask*)

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.
mask	The mask of the interrupt source to be set. Use the defined edma_interrupt_enable_t type.

12.7.16 void EDMA_TcdReset (edma_tcd_t * *tcd*)

This function sets all fields for this TCD structure to default value.

Parameters

tcd	Pointer to the TCD structure.

Note

This function enables the auto stop request feature.

12.7.17 void EDMA_TcdSetTransferConfig (edma_tcd_t * *tcd,* const edma_transfer_config_t * *config,* edma_tcd_t * *nextTcd*)

The TCD is a transfer control descriptor. The content of the TCD is the same as the hardware TCD registers. The 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

tcd	Pointer to the TCD structure.
config	Pointer to eDMA transfer configuration structure.
nextTcd	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.

12.7.18 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

tcd	A point to the TCD structure.
config	A pointer to the minor offset configuration structure.

12.7.19 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

tcd	Point to the TCD structure.
type	Channel link type, it can be one of: • kEDMA_LinkNone • kEDMA_MinorLink • kEDMA_MajorLink
linkedChannel	The linked channel number.

12.7.20 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

tcd	A pointer to the TCD structure.
bandWidth	A bandwidth setting, which can be one of the following: • kEDMABandwidthStallNone • kEDMABandwidthStall4Cycle • kEDMABandwidthStall8Cycle

12.7.21 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

tcd	A pointer to the TCD structure.
srcModulo	A source modulo value.
destModulo	A destination modulo value.

12.7.22 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

tcd	A pointer to the TCD structure.
enable	The command to enable (true) or disable (false).

12.7.23 void EDMA_TcdEnableInterrupts (edma_tcd_t * *tcd,* uint32_t *mask*)

Parameters

tcd	Point to the TCD structure.
mask	The mask of interrupt source to be set. Users need to use the defined edma_interrupt-_enable_t type.

12.7.24 void EDMA_TcdDisableInterrupts (edma_tcd_t ∗ *tcd,* uint32_t *mask*)

Parameters

tcd	Point to the TCD structure.
mask	The mask of interrupt source to be set. Users need to use the defined edma_interrupt-_enable_t type.

12.7.25 static void EDMA_EnableChannelRequest (DMA_Type ∗ *base,* uint32_t *channel*) `[inline], [static]`

This function enables the hardware channel request.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

12.7.26 static void EDMA_DisableChannelRequest (DMA_Type ∗ *base,* uint32_t *channel*) `[inline], [static]`

This function disables the hardware channel request.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

12.7.27 static void EDMA_TriggerChannelStart (DMA_Type * *base,* uint32_t *channel*) [inline], [static]

This function starts a minor loop transfer.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

12.7.28 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 the number of major loop count that has not finished.

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.

Returns

Major loop count which has not been transferred yet for the current TCD.

Note

1. This function can only be used to get unfinished major loop count of transfer without the next TCD, or it might be inaccuracy.
   1. The unfinished/remaining transfer bytes cannot be obtained directly from registers while the channel is running. Because to calculate the remaining bytes, the initial NBYTES configured in DMA_TCDn_NBYTES_MLNO register is needed while the eDMA IP does not support getting it while a channel is active. In another word, the NBYTES value reading is always the actual (decrementing) NBYTES value the dma_engine is working with while a channel is running. Consequently, to get the remaining transfer bytes, a software-saved initial value of NBYTES (for example copied before enabling the channel) is needed. The formula to calculate it is shown below: RemainingBytes = RemainingMajorLoopCount * NBYTES(initially configured)

12.7.29 static uint32_t EDMA_GetErrorStatusFlags (DMA_Type * *base*) [inline], [static]

Parameters

base	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.

12.7.30 uint32_t EDMA_GetChannelStatusFlags (DMA_Type * *base,* uint32_t *channel*)

Parameters

base	eDMA peripheral base address.
channel	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.

12.7.31 void EDMA_ClearChannelStatusFlags (DMA_Type * *base,* uint32_t *channel,* uint32_t *mask*)

Parameters

base	eDMA peripheral base address.
channel	eDMA channel number.
mask	The mask of channel status to be cleared. Users need to use the defined _edma_-channel_status_flags type.

12.7.32 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

handle	eDMA handle pointer. The eDMA handle stores callback function and parameters.
base	eDMA peripheral base address.
channel	eDMA channel number.

12.7.33 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.

Parameters

handle	eDMA handle pointer.
tcdPool	A memory pool to store TCDs. It must be 32 bytes aligned.
tcdSize	The number of TCD slots.

12.7.34 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.

Parameters

handle	eDMA handle pointer.
callback	eDMA callback function pointer.
userData	A parameter for the callback function.

12.7.35 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

config	The user configuration structure of type edma_transfer_t.
srcAddr	eDMA transfer source address.
srcWidth	eDMA transfer source address width(bytes).
destAddr	eDMA transfer destination address.
destWidth	eDMA transfer destination address width(bytes).
bytesEach-Request	eDMA transfer bytes per channel request.
transferBytes	eDMA transfer bytes to be transferred.
type	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).

12.7.36 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. If submitting the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

handle	eDMA handle pointer.
config	Pointer to eDMA transfer configuration structure.

Return values

kStatus_EDMA_Success	It means submit transfer request succeed.
kStatus_EDMA_Queue-Full	It means TCD queue is full. Submit transfer request is not allowed.

kStatus_EDMA_Busy	It means the given channel is busy, need to submit request later.

12.7.37 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

handle	eDMA handle pointer.

12.7.38 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

handle	eDMA handle pointer.

12.7.39 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

handle	DMA handle pointer.

12.7.40 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 part of the Reference Manual for further details.

Parameters

handle	eDMA handle pointer.

Chapter 13
C90TFS Flash Driver

13.1 Overview

The flash provides the C90TFS Flash driver of Kinetis devices with the C90TFS Flash module inside. The flash driver provides general APIs to handle specific operations on C90TFS/FTFx Flash module. The user can use those APIs directly in the application. In addition, it provides internal functions called by the driver. Although these functions are not meant to be called from the user's application directly, the APIs can still be used.

Data Structures

- struct flash_execute_in_ram_function_config_t
  Flash execute-in-RAM function information. More...
- struct flash_swap_state_config_t
  Flash Swap information. More...
- struct flash_swap_ifr_field_config_t
  Flash Swap IFR fields. More...
- union flash_swap_ifr_field_data_t
  Flash Swap IFR field data. More...
- union pflash_protection_status_low_t
  PFlash protection status - low 32bit. More...
- struct pflash_protection_status_t
  PFlash protection status - full. More...
- struct flash_prefetch_speculation_status_t
  Flash prefetch speculation status. More...
- struct flash_protection_config_t
  Active flash protection information for the current operation. More...
- struct flash_access_config_t
  Active flash Execute-Only access information for the current operation. More...
- struct flash_operation_config_t
  Active flash information for the current operation. More...
- struct flash_config_t
  Flash driver state information. More...

Typedefs

- typedef void(∗ flash_callback_t)(void)
  A callback type used for the Pflash block.

Enumerations

- enum flash_margin_value_t {
  kFLASH_MarginValueNormal,
  kFLASH_MarginValueUser,
  kFLASH_MarginValueFactory,

kFLASH_MarginValueInvalid }

Enumeration for supported flash margin levels.

- enum flash_security_state_t {
  kFLASH_SecurityStateNotSecure,
  kFLASH_SecurityStateBackdoorEnabled,
  kFLASH_SecurityStateBackdoorDisabled }

  Enumeration for the three possible flash security states.
- enum flash_protection_state_t {
  kFLASH_ProtectionStateUnprotected,
  kFLASH_ProtectionStateProtected,
  kFLASH_ProtectionStateMixed }

  Enumeration for the three possible flash protection levels.
- enum flash_execute_only_access_state_t {
  kFLASH_AccessStateUnLimited,
  kFLASH_AccessStateExecuteOnly,
  kFLASH_AccessStateMixed }

  Enumeration for the three possible flash execute access levels.
- enum flash_property_tag_t {
  kFLASH_PropertyPflashSectorSize = 0x00U,
  kFLASH_PropertyPflashTotalSize = 0x01U,
  kFLASH_PropertyPflashBlockSize = 0x02U,
  kFLASH_PropertyPflashBlockCount = 0x03U,
  kFLASH_PropertyPflashBlockBaseAddr = 0x04U,
  kFLASH_PropertyPflashFacSupport = 0x05U,
  kFLASH_PropertyPflashAccessSegmentSize = 0x06U,
  kFLASH_PropertyPflashAccessSegmentCount = 0x07U,
  kFLASH_PropertyFlexRamBlockBaseAddr = 0x08U,
  kFLASH_PropertyFlexRamTotalSize = 0x09U,
  kFLASH_PropertyDflashSectorSize = 0x10U,
  kFLASH_PropertyDflashTotalSize = 0x11U,
  kFLASH_PropertyDflashBlockSize = 0x12U,
  kFLASH_PropertyDflashBlockCount = 0x13U,
  kFLASH_PropertyDflashBlockBaseAddr = 0x14U,
  kFLASH_PropertyEepromTotalSize = 0x15U,
  kFLASH_PropertyFlashMemoryIndex = 0x20U,
  kFLASH_PropertyFlashCacheControllerIndex = 0x21U }

  Enumeration for various flash properties.
- enum _flash_execute_in_ram_function_constants {
  kFLASH_ExecuteInRamFunctionMaxSizeInWords = 16U,
  kFLASH_ExecuteInRamFunctionTotalNum = 2U }

  Constants for execute-in-RAM flash function.
- enum flash_read_resource_option_t {
  kFLASH_ResourceOptionFlashIfr,
  kFLASH_ResourceOptionVersionId = 0x01U }

  Enumeration for the two possible options of flash read resource command.
- enum _flash_read_resource_range {

kFLASH_ResourceRangePflashIfrSizeInBytes = 256U,
kFLASH_ResourceRangeVersionIdSizeInBytes = 8U,
kFLASH_ResourceRangeVersionIdStart = 0x00U,
kFLASH_ResourceRangeVersionIdEnd = 0x07U,
kFLASH_ResourceRangePflashSwapIfrStart = 0x20000U,
kFLASH_ResourceRangePflashSwapIfrEnd,
kFLASH_ResourceRangeDflashIfrStart = 0x800000U,
kFLASH_ResourceRangeDflashIfrEnd = 0x8003FFU }

Enumeration for the range of special-purpose flash resource.

- enum _k3_flash_read_once_index {
  kFLASH_RecordIndexSwapAddr = 0xA1U,
  kFLASH_RecordIndexSwapEnable = 0xA2U,
  kFLASH_RecordIndexSwapDisable = 0xA3U }

  Enumeration for the index of read/program once record.
- enum flash_flexram_function_option_t {
  kFLASH_FlexramFunctionOptionAvailableAsRam = 0xFFU,
  kFLASH_FlexramFunctionOptionAvailableForEeprom = 0x00U }

  Enumeration for the two possilbe options of set FlexRAM function command.
- enum _flash_acceleration_ram_property

  Enumeration for acceleration RAM property.
- enum flash_swap_function_option_t {
  kFLASH_SwapFunctionOptionEnable = 0x00U,
  kFLASH_SwapFunctionOptionDisable = 0x01U }

  Enumeration for the possible options of Swap function.
- enum flash_swap_control_option_t {
  kFLASH_SwapControlOptionIntializeSystem = 0x01U,
  kFLASH_SwapControlOptionSetInUpdateState = 0x02U,
  kFLASH_SwapControlOptionSetInCompleteState = 0x04U,
  kFLASH_SwapControlOptionReportStatus = 0x08U,
  kFLASH_SwapControlOptionDisableSystem = 0x10U }

  Enumeration for the possible options of Swap control commands.
- enum flash_swap_state_t {
  kFLASH_SwapStateUninitialized = 0x00U,
  kFLASH_SwapStateReady = 0x01U,
  kFLASH_SwapStateUpdate = 0x02U,
  kFLASH_SwapStateUpdateErased = 0x03U,
  kFLASH_SwapStateComplete = 0x04U,
  kFLASH_SwapStateDisabled = 0x05U }

  Enumeration for the possible flash Swap status.
- enum flash_swap_block_status_t {
  kFLASH_SwapBlockStatusLowerHalfProgramBlocksAtZero,
  kFLASH_SwapBlockStatusUpperHalfProgramBlocksAtZero }

  Enumeration for the possible flash Swap block status
- enum flash_partition_flexram_load_option_t {
  kFLASH_PartitionFlexramLoadOptionLoadedWithValidEepromData,
  kFLASH_PartitionFlexramLoadOptionNotLoaded = 0x01U }

  Enumeration for the FlexRAM load during reset option.

- enum flash_memory_index_t {
  kFLASH_MemoryIndexPrimaryFlash = 0x00U,
  kFLASH_MemoryIndexSecondaryFlash = 0x01U }
  Enumeration for the flash memory index.
- enum flash_cache_controller_index_t {
  kFLASH_CacheControllerIndexForCore0 = 0x00U,
  kFLASH_CacheControllerIndexForCore1 = 0x01U }
  Enumeration for the flash cache controller index.
- enum flash_prefetch_speculation_option_t
  Enumeration for the two possible options of flash prefetch speculation.
- enum flash_cache_clear_process_t {
  kFLASH_CacheClearProcessPre = 0x00U,
  kFLASH_CacheClearProcessPost = 0x01U }
  Flash cache clear process code.

Flash version

- enum _flash_driver_version_constants {
  kFLASH_DriverVersionName = 'F',
  kFLASH_DriverVersionMajor = 2,
  kFLASH_DriverVersionMinor = 3,
  kFLASH_DriverVersionBugfix = 1 }
  Flash driver version for ROM.
- #define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
  Constructs the version number for drivers.
- #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))
  Flash driver version for SDK.

Flash configuration

- #define FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT 1
  Indicates whether to support FlexNVM in the Flash driver.
- #define FLASH_SSD_IS_FLEXNVM_ENABLED (FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT && FSL_FEATURE_FLASH_HAS_FLEX_NVM)
  Indicates whether the FlexNVM is enabled in the Flash driver.
- #define FLASH_SSD_CONFIG_ENABLE_SECONDARY_FLASH_SUPPORT 1
  Indicates whether to support Secondary flash in the Flash driver.
- #define FLASH_SSD_IS_SECONDARY_FLASH_ENABLED (0)
  Indicates whether the secondary flash is supported in the Flash driver.
- #define FLASH_DRIVER_IS_FLASH_RESIDENT 1
  Flash driver location.
- #define FLASH_DRIVER_IS_EXPORTED 0
  Flash Driver Export option.

Flash status

- enum _flash_status {
  kStatus_FLASH_Success = MAKE_STATUS(kStatusGroupGeneric, 0),
  kStatus_FLASH_InvalidArgument = MAKE_STATUS(kStatusGroupGeneric, 4),
  kStatus_FLASH_SizeError = MAKE_STATUS(kStatusGroupFlashDriver, 0),
  kStatus_FLASH_AlignmentError,
  kStatus_FLASH_AddressError = MAKE_STATUS(kStatusGroupFlashDriver, 2),
  kStatus_FLASH_AccessError,
  kStatus_FLASH_ProtectionViolation,
  kStatus_FLASH_CommandFailure,
  kStatus_FLASH_UnknownProperty = MAKE_STATUS(kStatusGroupFlashDriver, 6),
  kStatus_FLASH_EraseKeyError = MAKE_STATUS(kStatusGroupFlashDriver, 7),
  kStatus_FLASH_RegionExecuteOnly,
  kStatus_FLASH_ExecuteInRamFunctionNotReady,
  kStatus_FLASH_PartitionStatusUpdateFailure,
  kStatus_FLASH_SetFlexramAsEepromError,
  kStatus_FLASH_RecoverFlexramAsRamError,
  kStatus_FLASH_SetFlexramAsRamError = MAKE_STATUS(kStatusGroupFlashDriver, 13),
  kStatus_FLASH_RecoverFlexramAsEepromError,
  kStatus_FLASH_CommandNotSupported = MAKE_STATUS(kStatusGroupFlashDriver, 15),
  kStatus_FLASH_SwapSystemNotInUninitialized,
  kStatus_FLASH_SwapIndicatorAddressError,
  kStatus_FLASH_ReadOnlyProperty = MAKE_STATUS(kStatusGroupFlashDriver, 18),
  kStatus_FLASH_InvalidPropertyValue,
  kStatus_FLASH_InvalidSpeculationOption }
  Flash driver status codes.
- #define kStatusGroupGeneric 0
  Flash driver status group.
- #define **kStatusGroupFlashDriver** 1
- #define MAKE_STATUS(group, code) ((((group)∗100) + (code)))
  Constructs a status code value from a group and a code number.

Flash API key

- enum _flash_driver_api_keys { kFLASH_ApiEraseKey = FOUR_CHAR_CODE('k', 'f', 'e', 'k') }
  Enumeration for Flash driver API keys.
- #define FOUR_CHAR_CODE(a, b, c, d) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))
  Constructs the four character code for the Flash driver API key.

Initialization

- status_t FLASH_Init (flash_config_t ∗config)
  Initializes the global flash properties structure members.
- status_t FLASH_SetCallback (flash_config_t ∗config, flash_callback_t callback)
  Sets the desired flash callback function.
- status_t FLASH_PrepareExecuteInRamFunctions (flash_config_t ∗config)
  Prepares flash execute-in-RAM functions.

Erasing

- status_t FLASH_EraseAll (flash_config_t *config, uint32_t key)
  Erases entire flash.
- status_t FLASH_Erase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, uint32_t key)
  Erases the flash sectors encompassed by parameters passed into function.
- status_t FLASH_EraseAllUnsecure (flash_config_t *config, uint32_t key)
  Erases the entire flash, including protected sectors.
- status_t FLASH_EraseAllExecuteOnlySegments (flash_config_t *config, uint32_t key)
  Erases all program flash execute-only segments defined by the FXACC registers.

Programming

- status_t FLASH_Program (flash_config_t *config, uint32_t start, uint32_t *src, uint32_t lengthInBytes)
  Programs flash with data at locations passed in through parameters.
- status_t FLASH_ProgramOnce (flash_config_t *config, uint32_t index, uint32_t *src, uint32_t lengthInBytes)
  Programs Program Once Field through parameters.

Reading

Programs flash with data at locations passed in through parameters via the Program Section command.

This function programs the flash memory with the desired data for a given flash area as determined by the start address and length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be programmed. Must be word-aligned.
src	A pointer to the source buffer of data that is to be programmed into the flash.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

kStatus_FLASH_AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FLASH_AddressError	Address is out of range.
kStatus_FLASH_SetFlexramAsRamError	Failed to set flexram as RAM.
kStatus_FLASH_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_AccessError	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_CommandFailure	Run-time error during command execution.
kStatus_FLASH_RecoverFlexramAsEepromError	Failed to recover FlexRAM as EEPROM.

Programs the EEPROM with data at locations passed in through parameters.

This function programs the emulated EEPROM with the desired data for a given flash area as determined by the start address and length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be programmed. Must be word-aligned.
src	A pointer to the source buffer of data that is to be programmed into the flash.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_InvalidArgument	An invalid argument is provided.

kStatus_FLASH_Address-Error	Address is out of range.
kStatus_FLASH_Set-FlexramAsEepromError	Failed to set flexram as eeprom.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_Recover-FlexramAsRamError	Failed to recover the FlexRAM as RAM.

- status_t FLASH_ReadResource (flash_config_t *config, uint32_t start, uint32_t *dst, uint32_-t lengthInBytes, flash_read_resource_option_t option)
  Reads the resource with data at locations passed in through parameters.
- status_t FLASH_ReadOnce (flash_config_t *config, uint32_t index, uint32_t *dst, uint32_t length-InBytes)
  Reads the Program Once Field through parameters.

Security

- status_t FLASH_GetSecurityState (flash_config_t *config, flash_security_state_t *state)
  Returns the security state via the pointer passed into the function.
- status_t FLASH_SecurityBypass (flash_config_t *config, const uint8_t *backdoorKey)
  Allows users to bypass security with a backdoor key.

Verification

- status_t FLASH_VerifyEraseAll (flash_config_t *config, flash_margin_value_t margin)
  Verifies erasure of the entire flash at a specified margin level.
- status_t FLASH_VerifyErase (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash-_margin_value_t margin)
  Verifies an erasure of the desired flash area at a specified margin level.
- status_t FLASH_VerifyProgram (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, const uint32_t *expectedData, flash_margin_value_t margin, uint32_t *failedAddress, uint32_-t *failedData)
  Verifies programming of the desired flash area at a specified margin level.
- status_t FLASH_VerifyEraseAllExecuteOnlySegments (flash_config_t *config, flash_margin_-value_t margin)
  Verifies whether the program flash execute-only segments have been erased to the specified read margin level.

Protection

- status_t FLASH_IsProtected (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash_-protection_state_t *protection_state)
  Returns the protection state of the desired flash area via the pointer passed into the function.
- status_t FLASH_IsExecuteOnly (flash_config_t *config, uint32_t start, uint32_t lengthInBytes, flash_execute_only_access_state_t *access_state)
  Returns the access state of the desired flash area via the pointer passed into the function.

Properties

- status_t FLASH_GetProperty (flash_config_t *config, flash_property_tag_t whichProperty, uint32_t *value)
  Returns the desired flash property.
- status_t FLASH_SetProperty (flash_config_t *config, flash_property_tag_t whichProperty, uint32_t value)
  Sets the desired flash property.

Flash Protection Utilities

Prepares the FlexNVM block for use as data flash, EEPROM backup, or a combination of both and initializes the FlexRAM.

Parameters

config	Pointer to storage for the driver runtime state.
option	The option used to set FlexRAM load behavior during reset.
eepromDataSizeCode	Determines the amount of FlexRAM used in each of the available EEPROM subsystems.
flexnvmPartitionCode	Specifies how to split the FlexNVM block between data flash memory and EEPROM backup memory supporting EEPROM functions.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_InvalidArgument	Invalid argument is provided.
kStatus_FLASH_ExecuteInRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_AccessError	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_CommandFailure	Run-time error during command execution.

- status_t FLASH_PflashSetProtection (flash_config_t *config, pflash_protection_status_t *protectStatus)
  Sets the PFlash Protection to the intended protection status.
- status_t FLASH_PflashGetProtection (flash_config_t *config, pflash_protection_status_t *protectStatus)
  Gets the PFlash protection status.

13.2 Data Structure Documentation

13.2.1 struct flash_execute_in_ram_function_config_t

Data Fields

- uint32_t activeFunctionCount
  Number of available execute-in-RAM functions.
- uint32_t ∗ flashRunCommand
  Execute-in-RAM function: flash_run_command.
- uint32_t ∗ flashCommonBitOperation
  Execute-in-RAM function: flash_common_bit_operation.

13.2.1.0.0.30 Field Documentation

13.2.1.0.0.30.1 uint32_t flash_execute_in_ram_function_config_t::activeFunctionCount

13.2.1.0.0.30.2 uint32_t∗ flash_execute_in_ram_function_config_t::flashRunCommand

13.2.1.0.0.30.3 uint32_t∗ flash_execute_in_ram_function_config_t::flashCommonBitOperation

13.2.2 struct flash_swap_state_config_t

Data Fields

- flash_swap_state_t flashSwapState
  The current Swap system status.
- flash_swap_block_status_t currentSwapBlockStatus
  The current Swap block status.
- flash_swap_block_status_t nextSwapBlockStatus
  The next Swap block status.

13.2.2.0.0.31 Field Documentation

13.2.2.0.0.31.1 flash_swap_state_t flash_swap_state_config_t::flashSwapState

13.2.2.0.0.31.2 flash_swap_block_status_t flash_swap_state_config_t::currentSwapBlockStatus

13.2.2.0.0.31.3 flash_swap_block_status_t flash_swap_state_config_t::nextSwapBlockStatus

13.2.3 struct flash_swap_ifr_field_config_t

Data Fields

- uint16_t swapIndicatorAddress
  A Swap indicator address field.
- uint16_t swapEnableWord
  A Swap enable word field.
- uint8_t reserved0 [4]

A reserved field.

13.2.3.0.0.32 Field Documentation

13.2.3.0.0.32.1 uint16_t flash_swap_ifr_field_config_t::swapIndicatorAddress

13.2.3.0.0.32.2 uint16_t flash_swap_ifr_field_config_t::swapEnableWord

13.2.3.0.0.32.3 uint8_t flash_swap_ifr_field_config_t::reserved0[4]

13.2.4 union flash_swap_ifr_field_data_t

Data Fields

- uint32_t flashSwapIfrData [2]
  A flash Swap IFR field data .
- flash_swap_ifr_field_config_t flashSwapIfrField
  A flash Swap IFR field structure.

13.2.4.0.0.33 Field Documentation

13.2.4.0.0.33.1 uint32_t flash_swap_ifr_field_data_t::flashSwapIfrData[2]

13.2.4.0.0.33.2 flash_swap_ifr_field_config_t flash_swap_ifr_field_data_t::flashSwapIfrField

13.2.5 union pflash_protection_status_low_t

Data Fields

- uint32_t protl32b
  PROT[31:0] .
- uint8_t protsl
  PROTS[7:0] .
- uint8_t protsh
  PROTS[15:8] .

13.2.5.0.0.34 Field Documentation

13.2.5.0.0.34.1 uint32_t pflash_protection_status_low_t::protl32b

13.2.5.0.0.34.2 uint8_t pflash_protection_status_low_t::protsl

13.2.5.0.0.34.3 uint8_t pflash_protection_status_low_t::protsh

13.2.6 struct pflash_protection_status_t

Data Fields

- pflash_protection_status_low_t valueLow32b
  PROT[31:0] or PROTS[15:0].

13.2.6.0.0.35 Field Documentation

13.2.6.0.0.35.1 pflash_protection_status_low_t pflash_protection_status_t::valueLow32b

13.2.7 struct flash_prefetch_speculation_status_t

Data Fields

- flash_prefetch_speculation_option_t instructionOption
  Instruction speculation.
- flash_prefetch_speculation_option_t dataOption
  Data speculation.

13.2.7.0.0.36 Field Documentation

13.2.7.0.0.36.1 flash_prefetch_speculation_option_t flash_prefetch_speculation_status_t-::instructionOption

13.2.7.0.0.36.2 flash_prefetch_speculation_option_t flash_prefetch_speculation_status_t::data-Option

13.2.8 struct flash_protection_config_t

Data Fields

- uint32_t regionBase
  Base address of flash protection region.
- uint32_t regionSize
  size of flash protection region.
- uint32_t regionCount
  flash protection region count.

13.2.8.0.0.37 Field Documentation

13.2.8.0.0.37.1 uint32_t flash_protection_config_t::regionBase

13.2.8.0.0.37.2 uint32_t flash_protection_config_t::regionSize

13.2.8.0.0.37.3 uint32_t flash_protection_config_t::regionCount

13.2.9 struct flash_access_config_t

Data Fields

- uint32_t SegmentBase
  Base address of flash Execute-Only segment.
- uint32_t SegmentSize
  size of flash Execute-Only segment.
- uint32_t SegmentCount
  flash Execute-Only segment count.

13.2.9.0.0.38 Field Documentation

13.2.9.0.0.38.1 uint32_t flash_access_config_t::SegmentBase

13.2.9.0.0.38.2 uint32_t flash_access_config_t::SegmentSize

13.2.9.0.0.38.3 uint32_t flash_access_config_t::SegmentCount

13.2.10 struct flash_operation_config_t

Data Fields

- uint32_t convertedAddress
  A converted address for the current flash type.
- uint32_t activeSectorSize
  A sector size of the current flash type.
- uint32_t activeBlockSize
  A block size of the current flash type.
- uint32_t blockWriteUnitSize
  The write unit size.
- uint32_t sectorCmdAddressAligment
  An erase sector command address alignment.
- uint32_t partCmdAddressAligment
  A program/verify part command address alignment.
- 32_t resourceCmdAddressAligment
  A read resource command address alignment.
- uint32_t checkCmdAddressAligment
  A program check command address alignment.

13.2.10.0.0.39 Field Documentation

13.2.10.0.0.39.1 uint32_t flash_operation_config_t::convertedAddress

13.2.10.0.0.39.2 uint32_t flash_operation_config_t::activeSectorSize

13.2.10.0.0.39.3 uint32_t flash_operation_config_t::activeBlockSize

13.2.10.0.0.39.4 uint32_t flash_operation_config_t::blockWriteUnitSize

13.2.10.0.0.39.5 uint32_t flash_operation_config_t::sectorCmdAddressAligment

13.2.10.0.0.39.6 uint32_t flash_operation_config_t::partCmdAddressAligment

13.2.10.0.0.39.7 uint32_t flash_operation_config_t::resourceCmdAddressAligment

13.2.10.0.0.39.8 uint32_t flash_operation_config_t::checkCmdAddressAligment

13.2.11 struct flash_config_t

An instance of this structure is allocated by the user of the flash driver and passed into each of the driver APIs.

Data Fields

- uint32_t PFlashBlockBase
  A base address of the first PFlash block.
- uint32_t PFlashTotalSize
  The size of the combined PFlash block.
- uint8_t PFlashBlockCount
  A number of PFlash blocks.
- uint8_t FlashMemoryIndex
  0 - primary flash; 1 - secondary flash
- uint8_t FlashCacheControllerIndex
  0 - Controller for core 0; 1 - Controller for core 1
- uint8_t Reserved0
  Reserved field 0.
- uint32_t PFlashSectorSize
  The size in bytes of a sector of PFlash.
- flash_callback_t PFlashCallback
  The callback function for the flash API.
- uint32_t PFlashAccessSegmentSize
  A size in bytes of an access segment of PFlash.
- uint32_t PFlashAccessSegmentCount
  A number of PFlash access segments.
- uint32_t * flashExecuteInRamFunctionInfo
  An information structure of the flash execute-in-RAM function.
- uint32_t FlexRAMBlockBase
  For the FlexNVM device, this is the base address of the FlexRAM.

- uint32_t FlexRAMTotalSize
  For the FlexNVM device, this is the size of the FlexRAM.
- uint32_t DFlashBlockBase
  For the FlexNVM device, this is the base address of the D-Flash memory (FlexNVM memory)
- uint32_t DFlashTotalSize
  For the FlexNVM device, this is the total size of the FlexNVM memory;.
- uint32_t EEpromTotalSize
  For the FlexNVM device, this is the size in bytes of the EEPROM area which was partitioned from FlexRAM.

13.2.11.0.0.40 Field Documentation

13.2.11.0.0.40.1 uint32_t flash_config_t::PFlashTotalSize

13.2.11.0.0.40.2 uint8_t flash_config_t::PFlashBlockCount

13.2.11.0.0.40.3 uint32_t flash_config_t::PFlashSectorSize

13.2.11.0.0.40.4 flash_callback_t flash_config_t::PFlashCallback

13.2.11.0.0.40.5 uint32_t flash_config_t::PFlashAccessSegmentSize

13.2.11.0.0.40.6 uint32_t flash_config_t::PFlashAccessSegmentCount

13.2.11.0.0.40.7 uint32_t* flash_config_t::flashExecuteInRamFunctionInfo

13.2.11.0.0.40.8 uint32_t flash_config_t::FlexRAMBlockBase

For the non-FlexNVM device, this is the base address of the acceleration RAM memory

13.2.11.0.0.40.9 uint32_t flash_config_t::FlexRAMTotalSize

For the non-FlexNVM device, this is the size of the acceleration RAM memory

13.2.11.0.0.40.10 uint32_t flash_config_t::DFlashBlockBase

For the non-FlexNVM device, this field is unused

13.2.11.0.0.40.11 uint32_t flash_config_t::DFlashTotalSize

For the non-FlexNVM device, this field is unused

13.2.11.0.0.40.12 uint32_t flash_config_t::EEpromTotalSize

For the non-FlexNVM device, this field is unused

13.3 Macro Definition Documentation

13.3.1 #define MAKE_VERSION(*major, minor, bugfix*) (((major) << 16) | ((minor) << 8) | (bugfix))

13.3.2 #define FSL_FLASH_DRIVER_VERSION (MAKE_VERSION(2, 3, 1))

Version 2.3.1.

13.3.3 #define FLASH_SSD_CONFIG_ENABLE_FLEXNVM_SUPPORT 1

Enables the FlexNVM support by default.

13.3.4 #define FLASH_SSD_CONFIG_ENABLE_SECONDARY_FLASH_SUPPORT 1

Enables the secondary flash support by default.

13.3.5 #define FLASH_DRIVER_IS_FLASH_RESIDENT 1

Used for the flash resident application.

13.3.6 #define FLASH_DRIVER_IS_EXPORTED 0

Used for the MCUXpresso SDK application.

13.3.7 #define kStatusGroupGeneric 0

13.3.8 #define MAKE_STATUS(*group, code*) ((((group)*100) + (code)))

13.3.9 #define FOUR_CHAR_CODE(*a, b, c, d*) (((d) << 24) | ((c) << 16) | ((b) << 8) | ((a)))

13.4 Enumeration Type Documentation

13.4.1 enum _flash_driver_version_constants

Enumerator

kFLASH_DriverVersionName Flash driver version name.

kFLASH_DriverVersionMajor Major flash driver version.
kFLASH_DriverVersionMinor Minor flash driver version.
kFLASH_DriverVersionBugfix Bugfix for flash driver version.

13.4.2 enum _flash_status

Enumerator

kStatus_FLASH_Success API is executed successfully.
kStatus_FLASH_InvalidArgument Invalid argument.
kStatus_FLASH_SizeError Error size.
kStatus_FLASH_AlignmentError Parameter is not aligned with the specified baseline.
kStatus_FLASH_AddressError Address is out of range.
kStatus_FLASH_AccessError Invalid instruction codes and out-of bound addresses.
kStatus_FLASH_ProtectionViolation The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_CommandFailure Run-time error during command execution.
kStatus_FLASH_UnknownProperty Unknown property.
kStatus_FLASH_EraseKeyError API erase key is invalid.
kStatus_FLASH_RegionExecuteOnly The current region is execute-only.
kStatus_FLASH_ExecuteInRamFunctionNotReady Execute-in-RAM function is not available.
kStatus_FLASH_PartitionStatusUpdateFailure Failed to update partition status.
kStatus_FLASH_SetFlexramAsEepromError Failed to set FlexRAM as EEPROM.
kStatus_FLASH_RecoverFlexramAsRamError Failed to recover FlexRAM as RAM.
kStatus_FLASH_SetFlexramAsRamError Failed to set FlexRAM as RAM.
kStatus_FLASH_RecoverFlexramAsEepromError Failed to recover FlexRAM as EEPROM.
kStatus_FLASH_CommandNotSupported Flash API is not supported.
kStatus_FLASH_SwapSystemNotInUninitialized Swap system is not in an uninitialzed state.
kStatus_FLASH_SwapIndicatorAddressError The swap indicator address is invalid.
kStatus_FLASH_ReadOnlyProperty The flash property is read-only.
kStatus_FLASH_InvalidPropertyValue The flash property value is out of range.
kStatus_FLASH_InvalidSpeculationOption The option of flash prefetch speculation is invalid.

13.4.3 enum _flash_driver_api_keys

Note

The resulting value is built with a byte order such that the string being readable in expected order when viewed in a hex editor, if the value is treated as a 32-bit little endian value.

Enumerator

kFLASH_ApiEraseKey Key value used to validate all flash erase APIs.

13.4.4 enum flash_margin_value_t

Enumerator

kFLASH_MarginValueNormal Use the 'normal' read level for 1s.
kFLASH_MarginValueUser Apply the 'User' margin to the normal read-1 level.
kFLASH_MarginValueFactory Apply the 'Factory' margin to the normal read-1 level.
kFLASH_MarginValueInvalid Not real margin level, Used to determine the range of valid margin level.

13.4.5 enum flash_security_state_t

Enumerator

kFLASH_SecurityStateNotSecure Flash is not secure.
kFLASH_SecurityStateBackdoorEnabled Flash backdoor is enabled.
kFLASH_SecurityStateBackdoorDisabled Flash backdoor is disabled.

13.4.6 enum flash_protection_state_t

Enumerator

kFLASH_ProtectionStateUnprotected Flash region is not protected.
kFLASH_ProtectionStateProtected Flash region is protected.
kFLASH_ProtectionStateMixed Flash is mixed with protected and unprotected region.

13.4.7 enum flash_execute_only_access_state_t

Enumerator

kFLASH_AccessStateUnLimited Flash region is unlimited.
kFLASH_AccessStateExecuteOnly Flash region is execute only.
kFLASH_AccessStateMixed Flash is mixed with unlimited and execute only region.

13.4.8 enum flash_property_tag_t

Enumerator

kFLASH_PropertyPflashSectorSize Pflash sector size property.
kFLASH_PropertyPflashTotalSize Pflash total size property.

kFLASH_PropertyPflashBlockSize Pflash block size property.
kFLASH_PropertyPflashBlockCount Pflash block count property.
kFLASH_PropertyPflashBlockBaseAddr Pflash block base address property.
kFLASH_PropertyPflashFacSupport Pflash fac support property.
kFLASH_PropertyPflashAccessSegmentSize Pflash access segment size property.
kFLASH_PropertyPflashAccessSegmentCount Pflash access segment count property.
kFLASH_PropertyFlexRamBlockBaseAddr FlexRam block base address property.
kFLASH_PropertyFlexRamTotalSize FlexRam total size property.
kFLASH_PropertyDflashSectorSize Dflash sector size property.
kFLASH_PropertyDflashTotalSize Dflash total size property.
kFLASH_PropertyDflashBlockSize Dflash block size property.
kFLASH_PropertyDflashBlockCount Dflash block count property.
kFLASH_PropertyDflashBlockBaseAddr Dflash block base address property.
kFLASH_PropertyEepromTotalSize EEPROM total size property.
kFLASH_PropertyFlashMemoryIndex Flash memory index property.
kFLASH_PropertyFlashCacheControllerIndex Flash cache controller index property.

13.4.9 enum _flash_execute_in_ram_function_constants

Enumerator

kFLASH_ExecuteInRamFunctionMaxSizeInWords The maximum size of execute-in-RAM function.
kFLASH_ExecuteInRamFunctionTotalNum Total number of execute-in-RAM functions.

13.4.10 enum flash_read_resource_option_t

Enumerator

kFLASH_ResourceOptionFlashIfr Select code for Program flash 0 IFR, Program flash swap 0 IFR, Data flash 0 IFR.
kFLASH_ResourceOptionVersionId Select code for the version ID.

13.4.11 enum _flash_read_resource_range

Enumerator

kFLASH_ResourceRangePflashIfrSizeInBytes Pflash IFR size in byte.
kFLASH_ResourceRangeVersionIdSizeInBytes Version ID IFR size in byte.
kFLASH_ResourceRangeVersionIdStart Version ID IFR start address.
kFLASH_ResourceRangeVersionIdEnd Version ID IFR end address.

kFLASH_ResourceRangePflashSwapIfrStart Pflash swap IFR start address.
kFLASH_ResourceRangePflashSwapIfrEnd Pflash swap IFR end address.
kFLASH_ResourceRangeDflashIfrStart Dflash IFR start address.
kFLASH_ResourceRangeDflashIfrEnd Dflash IFR end address.

13.4.12 enum _k3_flash_read_once_index

Enumerator

kFLASH_RecordIndexSwapAddr Index of Swap indicator address.
kFLASH_RecordIndexSwapEnable Index of Swap system enable.
kFLASH_RecordIndexSwapDisable Index of Swap system disable.

13.4.13 enum flash_flexram_function_option_t

Enumerator

kFLASH_FlexramFunctionOptionAvailableAsRam An option used to make FlexRAM available as RAM.
kFLASH_FlexramFunctionOptionAvailableForEeprom An option used to make FlexRAM available for EEPROM.

13.4.14 enum flash_swap_function_option_t

Enumerator

kFLASH_SwapFunctionOptionEnable An option used to enable the Swap function.
kFLASH_SwapFunctionOptionDisable An option used to disable the Swap function.

13.4.15 enum flash_swap_control_option_t

Enumerator

kFLASH_SwapControlOptionIntializeSystem An option used to initialize the Swap system.
kFLASH_SwapControlOptionSetInUpdateState An option used to set the Swap in an update state.

kFLASH_SwapControlOptionSetInCompleteState An option used to set the Swap in a complete state.
kFLASH_SwapControlOptionReportStatus An option used to report the Swap status.
kFLASH_SwapControlOptionDisableSystem An option used to disable the Swap status.

13.4.16 enum flash_swap_state_t

Enumerator

kFLASH_SwapStateUninitialized Flash Swap system is in an uninitialized state.
kFLASH_SwapStateReady Flash Swap system is in a ready state.
kFLASH_SwapStateUpdate Flash Swap system is in an update state.
kFLASH_SwapStateUpdateErased Flash Swap system is in an updateErased state.
kFLASH_SwapStateComplete Flash Swap system is in a complete state.
kFLASH_SwapStateDisabled Flash Swap system is in a disabled state.

13.4.17 enum flash_swap_block_status_t

Enumerator

kFLASH_SwapBlockStatusLowerHalfProgramBlocksAtZero Swap block status is that lower half program block at zero.
kFLASH_SwapBlockStatusUpperHalfProgramBlocksAtZero Swap block status is that upper half program block at zero.

13.4.18 enum flash_partition_flexram_load_option_t

Enumerator

kFLASH_PartitionFlexramLoadOptionLoadedWithValidEepromData FlexRAM is loaded with valid EEPROM data during reset sequence.
kFLASH_PartitionFlexramLoadOptionNotLoaded FlexRAM is not loaded during reset sequence.

13.4.19 enum flash_memory_index_t

Enumerator

kFLASH_MemoryIndexPrimaryFlash Current flash memory is primary flash.
kFLASH_MemoryIndexSecondaryFlash Current flash memory is secondary flash.

13.4.20 enum flash_cache_controller_index_t

Enumerator

kFLASH_CacheControllerIndexForCore0 Current flash cache controller is for core 0.
kFLASH_CacheControllerIndexForCore1 Current flash cache controller is for core 1.

13.4.21 enum flash_cache_clear_process_t

Enumerator

kFLASH_CacheClearProcessPre Pre flash cache clear process.
kFLASH_CacheClearProcessPost Post flash cache clear process.

13.5 Function Documentation

13.5.1 status_t FLASH_Init (flash_config_t ∗ *config*)

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

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_-PartitionStatusUpdate-Failure	Failed to update the partition status.

13.5.2 status_t FLASH_SetCallback (flash_config_t ∗ *config,* flash_callback_t *callback*)

Parameters

config	Pointer to the storage for the driver runtime state.
callback	A callback function to be stored in the driver.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

13.5.3 status_t FLASH_PrepareExecuteInRamFunctions (flash_config_t * *config*)

Parameters

config	Pointer to the storage for the driver runtime state.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

13.5.4 status_t FLASH_EraseAll (flash_config_t * *config*, uint32_t *key*)

Parameters

config	Pointer to the storage for the driver runtime state.
key	A value used to validate all flash erase APIs.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Erase-KeyError	API erase key is invalid.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.

kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during command execution.
kStatus_FLASH_-PartitionStatusUpdate-Failure	Failed to update the partition status.

13.5.5 status_t FLASH_Erase (flash_config_t ∗ *config*, uint32_t *start*, uint32_t *lengthInBytes*, uint32_t *key*)

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

Parameters

config	The pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be erased. The start address does not need to be sector-aligned but must be word-aligned.
lengthInBytes	The length, given in bytes (not words or long-words) to be erased. Must be word-aligned.
key	The value used to validate all flash erase APIs.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	The parameter is not aligned with the specified baseline.
kStatus_FLASH_Address-Error	The address is out of range.

kStatus_FLASH_Erase-KeyError	The API erase key is invalid.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.6 status_t FLASH_EraseAllUnsecure (flash_config_t * *config,* uint32_t *key*)

Parameters

config	Pointer to the storage for the driver runtime state.
key	A value used to validate all flash erase APIs.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Erase-KeyError	API erase key is invalid.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FLASH_-CommandFailure	Run-time error during command execution.
kStatus_FLASH_-PartitionStatusUpdate-Failure	Failed to update the partition status.

13.5.7 status_t FLASH_EraseAllExecuteOnlySegments (flash_config_t ∗ *config*, uint32_t *key*)

Parameters

config	Pointer to the storage for the driver runtime state.
key	A value used to validate all flash erase APIs.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Erase-KeyError	API erase key is invalid.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.8 status_t FLASH_Program (flash_config_t ∗ *config*, uint32_t *start*, uint32_t ∗ *src*, uint32_t *lengthInBytes*)

This function programs the flash memory with the desired data for a given flash area as determined by the start address and the length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be programmed. Must be word-aligned.
src	A pointer to the source buffer of data that is to be programmed into the flash.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FLASH_Address-Error	Address is out of range.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.9 status_t FLASH_ProgramOnce (flash_config_t ∗ *config*, uint32_t *index*, uint32_t ∗ *src*, uint32_t *lengthInBytes*)

This function programs the Program Once Field with the desired data for a given flash area as determined by the index and length.

Parameters

config	A pointer to the storage for the driver runtime state.
index	The index indicating which area of the Program Once Field to be programmed.
src	A pointer to the source buffer of data that is to be programmed into the Program Once Field.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.10 status_t FLASH_ReadResource (flash_config_t ∗ *config,* uint32_t *start,* uint32_t ∗ *dst,* uint32_t *lengthInBytes,* flash_read_resource_option_t *option*)

This function reads the flash memory with the desired location for a given flash area as determined by the start address and length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be programmed. Must be word-aligned.

dst	A pointer to the destination buffer of data that is used to store data to be read.
lengthInBytes	The length, given in bytes (not words or long-words), to be read. Must be word-aligned.
option	The resource option which indicates which area should be read back.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	Parameter is not aligned with the specified baseline.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.11 status_t FLASH_ReadOnce (flash_config_t ∗ *config*, uint32_t *index*, uint32_t ∗ *dst*, uint32_t *lengthInBytes*)

This function reads the read once feild with given index and length.

Parameters

config	A pointer to the storage for the driver runtime state.
index	The index indicating the area of program once field to be read.
dst	A pointer to the destination buffer of data that is used to store data to be read.
lengthInBytes	The length, given in bytes (not words or long-words), to be programmed. Must be word-aligned.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.12 status_t FLASH_GetSecurityState (flash_config_t * *config*, flash_security_state_t * *state*)

This function retrieves the current flash security status, including the security enabling state and the back-door key enabling state.

Parameters

config	A pointer to storage for the driver runtime state.
state	A pointer to the value returned for the current security status code:

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

13.5.13 status_t FLASH_SecurityBypass (flash_config_t * *config*, const uint8_t * *backdoorKey*)

If the MCU is in secured state, this function unsecures the MCU by comparing the provided backdoor key with ones in the flash configuration field.

Parameters

config	A pointer to the storage for the driver runtime state.
backdoorKey	A pointer to the user buffer containing the backdoor key.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.14 status_t FLASH_VerifyEraseAll (flash_config_t * *config*, flash_margin_value_t *margin*)

This function checks whether the flash is erased to the specified read margin level.

Parameters

config	A pointer to the storage for the driver runtime state.
margin	Read margin choice.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.15 status_t FLASH_VerifyErase (flash_config_t ∗ *config*, uint32_t *start*, uint32_t *lengthInBytes*, flash_margin_value_t *margin*)

This function checks the appropriate number of flash sectors based on the desired start address and length to check whether the flash is erased to the specified read margin level.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be verified. The start address does not need to be sector-aligned but must be word-aligned.
lengthInBytes	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
margin	Read margin choice.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FLASH_Address-Error	Address is out of range.

kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.16 status_t FLASH_VerifyProgram (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, const uint32_t * *expectedData*, flash_margin_value_t *margin*, uint32_t * *failedAddress*, uint32_t * *failedData*)

This function verifies the data programed in the flash memory using the Flash Program Check Command and compares it to the expected data for a given flash area as determined by the start address and length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be verified. Must be word-aligned.
lengthInBytes	The length, given in bytes (not words or long-words), to be verified. Must be word-aligned.
expectedData	A pointer to the expected data that is to be verified against.
margin	Read margin choice.
failedAddress	A pointer to the returned failing address.
failedData	A pointer to the returned failing data. Some derivatives do not include failed data as part of the FCCOBx registers. In this case, zeros are returned upon failure.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

kStatus_FLASH_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FLASH_Address-Error	Address is out of range.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.
kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.17 status_t FLASH_VerifyEraseAllExecuteOnlySegments (flash_config_t * *config*, flash_margin_value_t *margin*)

Parameters

config	A pointer to the storage for the driver runtime state.
margin	Read margin choice.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_Execute-InRamFunctionNotReady	Execute-in-RAM function is not available.
kStatus_FLASH_Access-Error	Invalid instruction codes and out-of bounds addresses.
kStatus_FLASH_-ProtectionViolation	The program/erase operation is requested to execute on protected areas.

kStatus_FLASH_-CommandFailure	Run-time error during the command execution.

13.5.18 status_t FLASH_IsProtected (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, flash_protection_state_t * *protection_state*)

This function retrieves the current flash protect status for a given flash area as determined by the start address and length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be checked. Must be word-aligned.
lengthInBytes	The length, given in bytes (not words or long-words) to be checked. Must be word-aligned.
protection_-state	A pointer to the value returned for the current protection status code for the desired flash area.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	Parameter is not aligned with specified baseline.
kStatus_FLASH_Address-Error	The address is out of range.

13.5.19 status_t FLASH_IsExecuteOnly (flash_config_t * *config*, uint32_t *start*, uint32_t *lengthInBytes*, flash_execute_only_access_state_t * *access_state*)

This function retrieves the current flash access status for a given flash area as determined by the start address and length.

Parameters

config	A pointer to the storage for the driver runtime state.
start	The start address of the desired flash memory to be checked. Must be word-aligned.
lengthInBytes	The length, given in bytes (not words or long-words), to be checked. Must be word-aligned.
access_state	A pointer to the value returned for the current access status code for the desired flash area.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-AlignmentError	The parameter is not aligned to the specified baseline.
kStatus_FLASH_Address-Error	The address is out of range.

13.5.20 status_t FLASH_GetProperty (flash_config_t * *config*, flash_property_tag_t *whichProperty*, uint32_t * *value*)

Parameters

config	A pointer to the storage for the driver runtime state.
whichProperty	The desired property from the list of properties in enum flash_property_tag_t
value	A pointer to the value returned for the desired flash property.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-UnknownProperty	An unknown property tag.

13.5.21 status_t FLASH_SetProperty (flash_config_t * *config*, flash_property_tag_t *whichProperty*, uint32_t *value*)

Parameters

config	A pointer to the storage for the driver runtime state.
whichProperty	The desired property from the list of properties in enum flash_property_tag_t
value	A to set for the desired flash property.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.
kStatus_FLASH_-UnknownProperty	An unknown property tag.
kStatus_FLASH_Invalid-PropertyValue	An invalid property value.
kStatus_FLASH_Read-OnlyProperty	An read-only property tag.

13.5.22 status_t FLASH_PflashSetProtection (flash_config_t ∗ *config*, pflash_protection_status_t ∗ *protectStatus*)

Parameters

config	A pointer to storage for the driver runtime state.
protectStatus	The expected protect status to set to the PFlash protection register. Each bit is corresponding to protection of 1/32(64) of the total PFlash. The least significant bit is corresponding to the lowest address area of PFlash. The most significant bit is corresponding to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

kStatus_FLASH_-CommandFailure	Run-time error during command execution.

13.5.23 status_t FLASH_PflashGetProtection (flash_config_t * *config*, pflash_protection_status_t * *protectStatus*)

Parameters

config	A pointer to the storage for the driver runtime state.
protectStatus	Protect status returned by the PFlash IP. Each bit is corresponding to the protection of 1/32(64) of the total PFlash. The least significant bit corresponds to the lowest address area of the PFlash. The most significant bit corresponds to the highest address area of PFlash. There are two possible cases as shown below: 0: this area is protected. 1: this area is unprotected.

Return values

kStatus_FLASH_Success	API was executed successfully.
kStatus_FLASH_Invalid-Argument	An invalid argument is provided.

Chapter 14
FlexCAN: Flex Controller Area Network Driver

14.1 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

14.2 FlexCAN Driver

14.2.1 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.

14.2.2 Typical use case

14.2.2.1 Message Buffer Send Operation

```
flexcan_config_t flexcanConfig;
flexcan_frame_t txFrame;

/* Init FlexCAN module. */
FLEXCAN_GetDefaultConfig(&flexcanConfig);
FLEXCAN_Init(EXAMPLE_CAN, &flexcanConfig);

/* Enable FlexCAN module. */
FLEXCAN_Enable(EXAMPLE_CAN, true);

/* Sets up the transmit message buffer. */
FLEXCAN_SetTxMbConfig(EXAMPLE_CAN, TX_MESSAGE_BUFFER_INDEX, true);

/* Prepares the transmit frame for sending. */
txFrame.format = KFLEXCAN_FrameFormatStandard;
txFrame.type   = KFLEXCAN_FrameTypeData;
txFrame.id     = FLEXCAN_ID_STD(0x123);
txFrame.length = 8;
txFrame.dataWord0 = CAN_WORD0_DATA_BYTE_0(0x11) |
                    CAN_WORD0_DATA_BYTE_1(0x22) |
                    CAN_WORD0_DATA_BYTE_2(0x33) |
                    CAN_WORD0_DATA_BYTE_3(0x44);
txFrame.dataWord1 = CAN_WORD1_DATA_BYTE_4(0x55) |
                    CAN_WORD1_DATA_BYTE_5(0x66) |
                    CAN_WORD1_DATA_BYTE_6(0x77) |
                    CAN_WORD1_DATA_BYTE_7(0x88);
/* Writes a transmit message buffer to send a CAN Message. */
FLEXCAN_WriteTxMb(EXAMPLE_CAN, TX_MESSAGE_BUFFER_INDEX, &txFrame);

/* Waits until the transmit message buffer is empty. */
while (!FLEXCAN_GetMbStatusFlags(EXAMPLE_CAN, 1 << TX_MESSAGE_BUFFER_INDEX));

/* Cleans the transmit message buffer empty status. */
FLEXCAN_ClearMbStatusFlags(EXAMPLE_CAN, 1 << TX_MESSAGE_BUFFER_INDEX);
```

14.2.2.2 Message Buffer Receive Operation

```
flexcan_config_t flexcanConfig;
flexcan_frame_t rxFrame;

/* Initializes the FlexCAN module. */
FLEXCAN_GetDefaultConfig(&flexcanConfig);
FLEXCAN_Init(EXAMPLE_CAN, &flexcanConfig);

/* Enables the FlexCAN module. */
FLEXCAN_Enable(EXAMPLE_CAN, true);

/* Sets up the receive message buffer. */
```

```
mbConfig.format    = KFLEXCAN_FrameFormatStandard;
mbConfig.type      = KFLEXCAN_FrameTypeData;
mbConfig.id        = FLEXCAN_ID_STD(0x123);
FLEXCAN_SetRxMbConfig(EXAMPLE_CAN, RX_MESSAGE_BUFFER_INDEX, &mbConfig, true);

/* Waits until the receive message buffer is full. */
while (!FLEXCAN_GetMbStatusFlags(EXAMPLE_CAN, 1 << RX_MESSAGE_BUFFER_INDEX));

/* Reads the received message from the receive message buffer. */
FLEXCAN_ReadRxMb(EXAMPLE_CAN, RX_MESSAGE_BUFFER_INDEX, &rxFrame);

/* Cleans the receive message buffer full status. */
FLEXCAN_ClearMbStatusFlags(EXAMPLE_CAN, 1 << RX_MESSAGE_BUFFER_INDEX);
```

14.2.2.3 Receive FIFO Operation

```
uint32_t rxFifoFilter[] = {FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x321, 0, 0),
                           FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x321, 1, 0),
                           FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x123, 0, 0),
                           FLEXCAN_RX_FIFO_STD_FILTER_TYPE_A(0x123, 1, 0)}
      ;

flexcan_config_t flexcanConfig;
flexcan_frame_t rxFrame;

/* Initializes the FlexCAN module. */
FLEXCAN_GetDefaultConfig(&flexcanConfig);
FLEXCAN_Init(EXAMPLE_CAN, &flexcanConfig);

/* Enables the FlexCAN module. */
FLEXCAN_Enable(EXAMPLE_CAN, true);

/* Sets up the receive FIFO. */
rxFifoConfig.idFilterTable = rxFifoFilter;
rxFifoConfig.idFilterType  = KFLEXCAN_RxFifoFilterTypeA;
rxFifoConfig.idFilterNum   = sizeof(rxFifoFilter) / sizeof(rxFifoFilter[0]);
rxFifoConfig.priority      = KFLEXCAN_RxFifoPrioHigh;
FlEXCAN_SetRxFifoConfig(EXAMPLE_CAN, &rxFifoConfig, true);

/* Waits until the receive FIFO becomes available. */
while (!FLEXCAN_GetMbStatusFlags(EXAMPLE_CAN, KFLEXCAN_RxFifoFrameAvlFlag));

/* Reads the message from the receive FIFO. */
FlEXCAN_ReadRxFifo(EXAMPLE_CAN, &rxFrame);

/* Cleans the receive FIFO available status. */
FLEXCAN_ClearMbStatusFlags(EXAMPLE_CAN, KFLEXCAN_RxFifoFrameAvlFlag);
```

Data Structures

- struct flexcan_frame_t
  FlexCAN message frame structure. More...
- struct flexcan_config_t
  FlexCAN module configuration structure. More...
- struct flexcan_timing_config_t
  FlexCAN protocol timing characteristic configuration structure. More...
- struct flexcan_rx_mb_config_t
  FlexCAN Receive Message Buffer configuration structure. More...
- struct flexcan_rx_fifo_config_t

FlexCAN 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 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 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) FLEXCAN_RX_FIFO_STD_MASK_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) FLEXCAN_RX_FIFO_EXT_MASK_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.

Typedefs

- typedef void(∗ flexcan_transfer_callback_t)(CAN_Type ∗base, flexcan_handle_t ∗handle, status_t status, uint32_t result, void ∗userData)
  FlexCAN transfer callback function.

Enumerations

- enum _flexcan_status {
  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_UnHandled = MAKE_STATUS(kStatusGroup_FLEXCAN, 11) }
  FlexCAN transfer status.
- enum flexcan_frame_format_t {
  kFLEXCAN_FrameFormatStandard = 0x0U,
  kFLEXCAN_FrameFormatExtend = 0x1U }
  FlexCAN frame format.
- enum flexcan_frame_type_t {
  kFLEXCAN_FrameTypeData = 0x0U,
  kFLEXCAN_FrameTypeRemote = 0x1U }
  FlexCAN frame type.
- enum flexcan_clock_source_t {
  kFLEXCAN_ClkSrcOsc = 0x0U,
  kFLEXCAN_ClkSrcPeri = 0x1U }
  FlexCAN clock 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_rx_fifo_priority_t {
  kFLEXCAN_RxFifoPrioLow = 0x0U,
  kFLEXCAN_RxFifoPrioHigh = 0x1U }
  FlexCAN Rx FIFO priority.
- enum _flexcan_interrupt_enable {
  kFLEXCAN_BusOffInterruptEnable = CAN_CTRL1_BOFFMSK_MASK,
  kFLEXCAN_ErrorInterruptEnable = CAN_CTRL1_ERRMSK_MASK,
  kFLEXCAN_RxWarningInterruptEnable = CAN_CTRL1_RWRNMSK_MASK,
  kFLEXCAN_TxWarningInterruptEnable = CAN_CTRL1_TWRNMSK_MASK,
  kFLEXCAN_WakeUpInterruptEnable = CAN_MCR_WAKMSK_MASK }
  FlexCAN interrupt configuration structure, default settings all disabled.
- enum _flexcan_flags {

kFLEXCAN_SynchFlag = CAN_ESR1_SYNCH_MASK,
kFLEXCAN_TxWarningIntFlag = CAN_ESR1_TWRNINT_MASK,
kFLEXCAN_RxWarningIntFlag = CAN_ESR1_RWRNINT_MASK,
kFLEXCAN_TxErrorWarningFlag = CAN_ESR1_TXWRN_MASK,
kFLEXCAN_RxErrorWarningFlag = CAN_ESR1_RXWRN_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_ErrorFlag }

FlexCAN status flags.

- enum _flexcan_error_flags {
  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 _flexcan_rx_fifo_flags {
  kFLEXCAN_RxFifoOverflowFlag = CAN_IFLAG1_BUF7I_MASK,
  kFLEXCAN_RxFifoWarningFlag = CAN_IFLAG1_BUF6I_MASK,
  kFLEXCAN_RxFifoFrameAvlFlag = CAN_IFLAG1_BUF5I_MASK }

  FlexCAN Rx FIFO status flags.

Driver version

- #define FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

  FlexCAN driver version 2.2.0.

Initialization and deinitialization

- void FLEXCAN_Init (CAN_Type *base, const flexcan_config_t *config, uint32_t sourceClock_Hz)

  Initializes a FlexCAN instance.
- void FLEXCAN_Deinit (CAN_Type *base)

  De-initializes a FlexCAN instance.
- void FLEXCAN_GetDefaultConfig (flexcan_config_t *config)

  Gets the default configuration structure.

Configuration.

- void FLEXCAN_SetTimingConfig (CAN_Type *base, const flexcan_timing_config_t *config)
  Sets the FlexCAN protocol 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_SetRxMbConfig (CAN_Type *base, uint8_t mbIdx, const flexcan_rx_mb_config_t *config, bool enable)
  Configures a FlexCAN Receive Message Buffer.
- void FLEXCAN_SetRxFifoConfig (CAN_Type *base, const flexcan_rx_fifo_config_t *config, bool enable)
  Configures the FlexCAN Rx FIFO.

Status

- static uint32_t FLEXCAN_GetStatusFlags (CAN_Type *base)
  Gets the FlexCAN module interrupt flags.
- static void FLEXCAN_ClearStatusFlags (CAN_Type *base, uint32_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 uint32_t FLEXCAN_GetMbStatusFlags (CAN_Type *base, uint32_t mask)
  Gets the FlexCAN Message Buffer interrupt flags.
- static void FLEXCAN_ClearMbStatusFlags (CAN_Type *base, uint32_t mask)
  Clears the FlexCAN Message Buffer interrupt flags.

Interrupts

- static void FLEXCAN_EnableInterrupts (CAN_Type *base, uint32_t mask)
  Enables FlexCAN interrupts according to the provided mask.
- static void FLEXCAN_DisableInterrupts (CAN_Type *base, uint32_t mask)
  Disables FlexCAN interrupts according to the provided mask.
- static void FLEXCAN_EnableMbInterrupts (CAN_Type *base, uint32_t mask)
  Enables FlexCAN Message Buffer interrupts.
- static void FLEXCAN_DisableMbInterrupts (CAN_Type *base, uint32_t mask)
  Disables FlexCAN Message Buffer interrupts.

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 *tx-Frame)
  Writes a FlexCAN Message to the Transmit Message Buffer.
- status_t FLEXCAN_ReadRxMb (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *rxFrame)
  Reads a FlexCAN Message from Receive Message Buffer.
- status_t FLEXCAN_ReadRxFifo (CAN_Type *base, flexcan_frame_t *rxFrame)
  Reads a FlexCAN Message from Rx FIFO.

Transactional

- status_t FLEXCAN_TransferSendBlocking (CAN_Type *base, uint8_t mbIdx, flexcan_frame_-t *txFrame)
  Performs a polling send transaction on the CAN bus.
- status_t FLEXCAN_TransferReceiveBlocking (CAN_Type *base, uint8_t mbIdx, flexcan_frame_t *rxFrame)
  Performs a polling receive transaction on the CAN bus.
- status_t FLEXCAN_TransferReceiveFifoBlocking (CAN_Type *base, flexcan_frame_t *rxFrame)
  Performs a polling receive transaction from 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 *xfer)
  Sends a message using IRQ.
- status_t FLEXCAN_TransferReceiveNonBlocking (CAN_Type *base, flexcan_handle_t *handle, flexcan_mb_transfer_t *xfer)
  Receives a message using IRQ.
- status_t FLEXCAN_TransferReceiveFifoNonBlocking (CAN_Type *base, flexcan_handle_-t *handle, flexcan_fifo_transfer_t *xfer)
  Receives a message from Rx FIFO using IRQ.
- 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.

14.2.3 Data Structure Documentation

14.2.3.1 struct flexcan_frame_t

14.2.3.1.0.41 Field Documentation

14.2.3.1.0.41.1 uint32_t flexcan_frame_t::timestamp

14.2.3.1.0.41.2 uint32_t flexcan_frame_t::length

14.2.3.1.0.41.3 uint32_t flexcan_frame_t::type

14.2.3.1.0.41.4 uint32_t flexcan_frame_t::format

14.2.3.1.0.41.5 uint32_t flexcan_frame_t::__pad0__

14.2.3.1.0.41.6 uint32_t flexcan_frame_t::idhit

14.2.3.1.0.41.7 uint32_t flexcan_frame_t::id

14.2.3.1.0.41.8 uint32_t flexcan_frame_t::dataWord0

14.2.3.1.0.41.9 uint32_t flexcan_frame_t::dataWord1

14.2.3.1.0.41.10 uint8_t flexcan_frame_t::dataByte3

14.2.3.1.0.41.11 uint8_t flexcan_frame_t::dataByte2

14.2.3.1.0.41.12 uint8_t flexcan_frame_t::dataByte1

14.2.3.1.0.41.13 uint8_t flexcan_frame_t::dataByte0

14.2.3.1.0.41.14 uint8_t flexcan_frame_t::dataByte7

14.2.3.1.0.41.15 uint8_t flexcan_frame_t::dataByte6

14.2.3.1.0.41.16 uint8_t flexcan_frame_t::dataByte5

14.2.3.1.0.41.17 uint8_t flexcan_frame_t::dataByte4

14.2.3.2 struct flexcan_config_t

Data Fields

- uint32_t baudRate
  FlexCAN baud rate in bps.
- flexcan_clock_source_t clkSrc
  Clock source for FlexCAN Protocol Engine.
- uint8_t maxMbNum
  The maximum number of Message Buffers used by user.

- bool enableLoopBack
  Enable or Disable Loop Back Self Test Mode.
- bool enableSelfWakeup
  Enable or Disable Self Wakeup Mode.
- bool enableIndividMask
  Enable or Disable Rx Individual Mask.

14.2.3.2.0.42 Field Documentation

14.2.3.2.0.42.1 uint32_t flexcan_config_t::baudRate

14.2.3.2.0.42.2 flexcan_clock_source_t flexcan_config_t::clkSrc

14.2.3.2.0.42.3 uint8_t flexcan_config_t::maxMbNum

14.2.3.2.0.42.4 bool flexcan_config_t::enableLoopBack

14.2.3.2.0.42.5 bool flexcan_config_t::enableSelfWakeup

14.2.3.2.0.42.6 bool flexcan_config_t::enableIndividMask

14.2.3.3 struct flexcan_timing_config_t

Data Fields

- uint16_t preDivider
  Clock Pre-scaler Division Factor.
- uint8_t rJumpwidth
  Re-sync Jump Width.
- uint8_t phaseSeg1
  Phase Segment 1.
- uint8_t phaseSeg2
  Phase Segment 2.
- uint8_t propSeg
  Propagation Segment.

14.2.3.3.0.43 Field Documentation

14.2.3.3.0.43.1 uint16_t flexcan_timing_config_t::preDivider

14.2.3.3.0.43.2 uint8_t flexcan_timing_config_t::rJumpwidth

14.2.3.3.0.43.3 uint8_t flexcan_timing_config_t::phaseSeg1

14.2.3.3.0.43.4 uint8_t flexcan_timing_config_t::phaseSeg2

14.2.3.3.0.43.5 uint8_t flexcan_timing_config_t::propSeg

14.2.3.4 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).

14.2.3.4.0.44 Field Documentation

14.2.3.4.0.44.1 uint32_t flexcan_rx_mb_config_t::id

14.2.3.4.0.44.2 flexcan_frame_format_t flexcan_rx_mb_config_t::format

14.2.3.4.0.44.3 flexcan_frame_type_t flexcan_rx_mb_config_t::type

14.2.3.5 struct flexcan_rx_fifo_config_t

Data Fields

- uint32_t * idFilterTable
  Pointer to the FlexCAN Rx FIFO identifier filter table.
- uint8_t idFilterNum
  The quantity of filter elements.
- flexcan_rx_fifo_filter_type_t idFilterType
  The FlexCAN Rx FIFO Filter type.
- flexcan_rx_fifo_priority_t priority
  The FlexCAN Rx FIFO receive priority.

14.2.3.5.0.45 Field Documentation

14.2.3.5.0.45.1 uint32_t* flexcan_rx_fifo_config_t::idFilterTable

14.2.3.5.0.45.2 uint8_t flexcan_rx_fifo_config_t::idFilterNum

14.2.3.5.0.45.3 flexcan_rx_fifo_filter_type_t flexcan_rx_fifo_config_t::idFilterType

14.2.3.5.0.45.4 flexcan_rx_fifo_priority_t flexcan_rx_fifo_config_t::priority

14.2.3.6 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.

14.2.3.6.0.46 Field Documentation

14.2.3.6.0.46.1 flexcan_frame_t* flexcan_mb_transfer_t::frame

14.2.3.6.0.46.2 uint8_t flexcan_mb_transfer_t::mbIdx

14.2.3.7 struct flexcan_fifo_transfer_t

Data Fields

- flexcan_frame_t * frame
  The buffer of CAN Message to be received from Rx FIFO.

14.2.3.7.0.47 Field Documentation

14.2.3.7.0.47.1 flexcan_frame_t* flexcan_fifo_transfer_t::frame

14.2.3.8 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 data from Message Buffers.
- flexcan_frame_t *volatile rxFifoFrameBuf
  The buffer for received data from Rx FIFO.

- volatile uint8_t mbState [CAN_WORD1_COUNT]
  Message Buffer transfer state.
- volatile uint8_t rxFifoState
  Rx FIFO transfer state.

14.2.3.8.0.48 Field Documentation

14.2.3.8.0.48.1 flexcan_transfer_callback_t flexcan_handle_t::callback

14.2.3.8.0.48.2 void∗ flexcan_handle_t::userData

14.2.3.8.0.48.3 flexcan_frame_t∗ volatile flexcan_handle_t::mbFrameBuf[CAN_WORD1_COUNT]

14.2.3.8.0.48.4 flexcan_frame_t∗ volatile flexcan_handle_t::rxFifoFrameBuf

14.2.3.8.0.48.5 volatile uint8_t flexcan_handle_t::mbState[CAN_WORD1_COUNT]

14.2.3.8.0.48.6 volatile uint8_t flexcan_handle_t::rxFifoState

14.2.4 Macro Definition Documentation

14.2.4.1 #define FLEXCAN_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

14.2.4.2 #define FLEXCAN_ID_STD(*id*) (((uint32_t)(((uint32_t)(id)) << CAN_ID_STD_SHIFT)) & CAN_ID_STD_MASK)

Standard Frame ID helper macro.

14.2.4.3 #define FLEXCAN_ID_EXT(*id*)

Value:

```
(((uint32_t)(((uint32_t)(id)) << CAN_ID_EXT_SHIFT)) & \
     (CAN_ID_EXT_MASK | CAN_ID_STD_MASK))
```

14.2.4.4 #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.

14.2.4.5 #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))
```

14.2.4.6 #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.

14.2.4.7 #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))
```

14.2.4.8 #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))
```

14.2.4.9 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(*id*) (((uint32_t)(id) & 0x7F8) << 21)

14.2.4.10 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(*id*) (((uint32_t)(id) & 0x7F8) << 13)

14.2.4.11 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(*id*) (((uint32_t)(id) & 0x7F8) << 5)

14.2.4.12 #define FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(*id*) (((uint32_t)(id) & 0x7F8) >> 3)

14.2.4.13 #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))
```

14.2.4.14 #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))
```

14.2.4.15 #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))
```

14.2.4.16 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(*id*) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) << 3)

14.2.4.17 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(*id*)

Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>            \
     5)
```

14.2.4.18 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(*id*)

Value:

```
((FLEXCAN_ID_EXT(id) & 0x1FE00000) >>           \
     13)
```

14.2.4.19 #define FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(*id*) ((FLEXCAN_ID_EXT(id) & 0x1FE00000) >> 21)

14.2.4.20 #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.

14.2.4.21 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_HIGH(*id, rtr, ide*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_HIGH(                    \
        id, rtr, ide)
```

14.2.4.22 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_B_LOW(*id, rtr, ide*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_B_LOW(                    \
        id, rtr, ide)
```

14.2.4.23 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_HIGH(          \
        id)
```

14.2.4.24 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_HIGH(          \
        id)
```

14.2.4.25 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_MID_LOW(*id*)

Value:

```
FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_MID_LOW(          \
        id)
```

14.2.4.26 #define FLEXCAN_RX_FIFO_STD_FILTER_TYPE_C_LOW(*id*) FLEXCAN_RX_FIFO_STD_MASK_TYPE_C_LOW(id)

\

14.2.4.27 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_A(*id, rtr, ide*) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_A(id, rtr, ide)

14.2.4.28 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_HIGH(*id, rtr, ide*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_HIGH(                    \
        id, rtr, ide)
```

14.2.4.29 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_B_LOW(*id, rtr, ide*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_B_LOW(                    \
        id, rtr, ide)
```

14.2.4.30 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_HIGH(*id*) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_HIGH(id)

\

14.2.4.31 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_HIGH(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_HIGH(           \
        id)
```

14.2.4.32 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_MID_LOW(*id*)

Value:

```
FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_MID_LOW(           \
        id)
```

14.2.4.33 #define FLEXCAN_RX_FIFO_EXT_FILTER_TYPE_C_LOW(*id*) FLEXCAN_RX_FIFO_EXT_MASK_TYPE_C_LOW(id)

14.2.5 Typedef Documentation

14.2.5.1 typedef void(∗ flexcan_transfer_callback_t)(CAN_Type ∗base, flexcan_handle_t ∗handle, status_t status, uint32_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.

14.2.6 Enumeration Type Documentation

14.2.6.1 enum _flexcan_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_UnHandled UnHadled Interrupt asserted.

14.2.6.2 enum flexcan_frame_format_t

Enumerator

kFLEXCAN_FrameFormatStandard Standard frame format attribute.
kFLEXCAN_FrameFormatExtend Extend frame format attribute.

14.2.6.3 enum flexcan_frame_type_t

Enumerator

kFLEXCAN_FrameTypeData Data frame type attribute.
kFLEXCAN_FrameTypeRemote Remote frame type attribute.

14.2.6.4 enum flexcan_clock_source_t

Enumerator

kFLEXCAN_ClkSrcOsc FlexCAN Protocol Engine clock from Oscillator.
kFLEXCAN_ClkSrcPeri FlexCAN Protocol Engine clock from Peripheral Clock.

14.2.6.5 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.

14.2.6.6 enum flexcan_rx_fifo_priority_t

The matching process starts from the Rx MB(or Rx FIFO) with higher priority. If no MB(or Rx FIFO filter) is satisfied, the matching process goes on with the 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 Rx FIFO first.

14.2.6.7 enum _flexcan_interrupt_enable

This structure contains the settings for all of the FlexCAN Module interrupt configurations. Note: FlexCAN Message Buffers and Rx FIFO have their own interrupts.

Enumerator

kFLEXCAN_BusOffInterruptEnable Bus Off interrupt.

kFLEXCAN_ErrorInterruptEnable Error interrupt.
kFLEXCAN_RxWarningInterruptEnable Rx Warning interrupt.
kFLEXCAN_TxWarningInterruptEnable Tx Warning interrupt.
kFLEXCAN_WakeUpInterruptEnable Wake Up interrupt.

14.2.6.8 enum _flexcan_flags

This provides constants for the FlexCAN status flags for use in the FlexCAN functions. Note: The CPU read action clears FlEXCAN_ErrorFlag, therefore user need to read FlEXCAN_ErrorFlag and distinguish which error is occur using _flexcan_error_flags enumerations.

Enumerator

kFLEXCAN_SynchFlag CAN Synchronization Status.
kFLEXCAN_TxWarningIntFlag Tx Warning Interrupt Flag.
kFLEXCAN_RxWarningIntFlag Rx Warning Interrupt Flag.
kFLEXCAN_TxErrorWarningFlag Tx Error Warning Status.
kFLEXCAN_RxErrorWarningFlag Rx Error Warning Status.
kFLEXCAN_IdleFlag CAN IDLE Status Flag.
kFLEXCAN_FaultConfinementFlag Fault Confinement State Flag.
kFLEXCAN_TransmittingFlag FlexCAN In Transmission Status.
kFLEXCAN_ReceivingFlag FlexCAN In Reception Status.
kFLEXCAN_BusOffIntFlag Bus Off Interrupt Flag.
kFLEXCAN_ErrorIntFlag Error Interrupt Flag.
kFLEXCAN_WakeUpIntFlag Wake-Up Interrupt Flag.
kFLEXCAN_ErrorFlag All FlexCAN Error Status.

14.2.6.9 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 ditermine which error is generated.

Enumerator

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.

14.2.6.10 enum _flexcan_rx_fifo_flags

The FlexCAN 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.

14.2.7 Function Documentation

14.2.7.1 void FLEXCAN_Init (CAN_Type ∗ *base,* const flexcan_config_t ∗ *config,* 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_ClkSrcOsc;
*   flexcanConfig.baudRate          = 125000U;
*   flexcanConfig.maxMbNum          = 16;
*   flexcanConfig.enableLoopBack    = false;
*   flexcanConfig.enableSelfWakeup  = false;
*   flexcanConfig.enableIndividMask = false;
*   flexcanConfig.enableDoze        = false;
*   FLEXCAN_Init(CAN0, &flexcanConfig, 8000000UL);
*
```

Parameters

base	FlexCAN peripheral base address.
config	Pointer to the user-defined configuration structure.
sourceClock_Hz	FlexCAN Protocol Engine clock source frequency in Hz.

14.2.7.2 void FLEXCAN_Deinit (CAN_Type ∗ *base*)

This function disables the FlexCAN module clock and sets all register values to the reset value.

Parameters

base	FlexCAN peripheral base address.

14.2.7.3 void FLEXCAN_GetDefaultConfig (flexcan_config_t * *config*)

This function initializes the FlexCAN configuration structure to default values. The default values are as follows. flexcanConfig->clkSrc = KFLEXCAN_ClkSrcOsc; flexcanConfig->baudRate = 125000U; flexcanConfig->maxMbNum = 16; flexcanConfig->enableLoopBack = false; flexcanConfig->enableSelfWakeup = false; flexcanConfig->enableIndividMask = false; flexcanConfig->enableDoze = false;

Parameters

config	Pointer to the FlexCAN configuration structure.

14.2.7.4 void FLEXCAN_SetTimingConfig (CAN_Type * *base*, const flexcan_timing_config_t * *config*)

This function gives user settings to CAN bus timing characteristic. The function is for an experienced user. For less experienced users, call the FLEXCAN_Init() and fill the baud rate field with a desired value. This provides the default timing characteristics to the module.

Note that calling FLEXCAN_SetTimingConfig() overrides the baud rate set in FLEXCAN_Init().

Parameters

base	FlexCAN peripheral base address.
config	Pointer to the timing configuration structure.

14.2.7.5 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

base	FlexCAN peripheral base address.

mask	Rx Message Buffer Global Mask value.

14.2.7.6 void FLEXCAN_SetRxFifoGlobalMask (CAN_Type ∗ *base,* uint32_t *mask*)

This function sets the global mask for FlexCAN FIFO in a matching process.

Parameters

base	FlexCAN peripheral base address.
mask	Rx Fifo Global Mask value.

14.2.7.7 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

base	FlexCAN peripheral base address.
maskIdx	The Index of individual Mask.
mask	Rx Individual Mask value.

14.2.7.8 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

base	FlexCAN peripheral base address.

mbIdx	The Message Buffer index.
enable	Enable/disable Tx Message Buffer. • true: Enable Tx Message Buffer. • false: Disable Tx Message Buffer.

14.2.7.9 void FLEXCAN_SetRxMbConfig (CAN_Type * *base*, uint8_t *mbIdx*, const flexcan_rx_mb_config_t * *config*, bool *enable*)

This function cleans a FlexCAN build-in Message Buffer and configures it as a Receive Message Buffer.

Parameters

base	FlexCAN peripheral base address.
mbIdx	The Message Buffer index.
config	Pointer to the FlexCAN Message Buffer configuration structure.
enable	Enable/disable Rx Message Buffer. • true: Enable Rx Message Buffer. • false: Disable Rx Message Buffer.

14.2.7.10 void FLEXCAN_SetRxFifoConfig (CAN_Type * *base*, const flexcan_rx_fifo_config_t * *config*, bool *enable*)

This function configures the Rx FIFO with given Rx FIFO configuration.

Parameters

base	FlexCAN peripheral base address.
config	Pointer to the FlexCAN Rx FIFO configuration structure.
enable	Enable/disable Rx FIFO. • true: Enable Rx FIFO. • false: Disable Rx FIFO.

14.2.7.11 static uint32_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

base	FlexCAN peripheral base address.

Returns

FlexCAN status flags which are ORed by the enumerators in the _flexcan_flags.

14.2.7.12 static void FLEXCAN_ClearStatusFlags (CAN_Type * *base*, uint32_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

base	FlexCAN peripheral base address.
mask	The status flags to be cleared, it is logical OR value of _flexcan_flags.

14.2.7.13 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

base	FlexCAN peripheral base address.
txErrBuf	Buffer to store Tx Error Counter value.
rxErrBuf	Buffer to store Rx Error Counter value.

14.2.7.14 static uint32_t FLEXCAN_GetMbStatusFlags (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function gets the interrupt flags of a given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

Returns

The status of given Message Buffers.

14.2.7.15 static void FLEXCAN_ClearMbStatusFlags (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function clears the interrupt flags of a given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

14.2.7.16 static void FLEXCAN_EnableInterrupts (CAN_Type * *base*, uint32_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

base	FlexCAN peripheral base address.
mask	The interrupts to enable. Logical OR of _flexcan_interrupt_enable.

14.2.7.17 static void FLEXCAN_DisableInterrupts (CAN_Type * *base*, uint32_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

base	FlexCAN peripheral base address.
mask	The interrupts to disable. Logical OR of _flexcan_interrupt_enable.

14.2.7.18 static void FLEXCAN_EnableMbInterrupts (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function enables the interrupts of given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

14.2.7.19 static void FLEXCAN_DisableMbInterrupts (CAN_Type * *base*, uint32_t *mask*) [inline], [static]

This function disables the interrupts of given Message Buffers.

Parameters

base	FlexCAN peripheral base address.
mask	The ORed FlexCAN Message Buffer mask.

14.2.7.20 static void FLEXCAN_Enable (CAN_Type * *base*, bool *enable*) [inline], [static]

This function enables or disables the FlexCAN module.

Parameters

base	FlexCAN base pointer.
enable	true to enable, false to disable.

14.2.7.21 status_t FLEXCAN_WriteTxMb (CAN_Type * *base*, uint8_t *mbIdx*, const flexcan_frame_t * *txFrame*)

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

base	FlexCAN peripheral base address.
mbIdx	The FlexCAN Message Buffer index.
txFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

14.2.7.22 status_t FLEXCAN_ReadRxMb (CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *rxFrame*)

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

base	FlexCAN peripheral base address.
mbIdx	The FlexCAN Message Buffer index.
rxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Rx Message Buffer is full and has been read successfully.
kStatus_FLEXCAN_RxOverflow	- Rx Message Buffer is already overflowed and has been read successfully.
kStatus_Fail	- Rx Message Buffer is empty.

14.2.7.23 status_t FLEXCAN_ReadRxFifo (CAN_Type * *base*, flexcan_frame_t * *rxFrame*)

This function reads a CAN message from the FlexCAN build-in Rx FIFO.

Parameters

base	FlexCAN peripheral base address.
rxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx FIFO successfully.
kStatus_Fail	- Rx FIFO is not enabled.

14.2.7.24 status_t FLEXCAN_TransferSendBlocking (CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *txFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	FlexCAN peripheral base pointer.
mbIdx	The FlexCAN Message Buffer index.
txFrame	Pointer to CAN message frame to be sent.

Return values

kStatus_Success	- Write Tx Message Buffer Successfully.
kStatus_Fail	- Tx Message Buffer is currently in use.

14.2.7.25 status_t FLEXCAN_TransferReceiveBlocking (CAN_Type * *base*, uint8_t *mbIdx*, flexcan_frame_t * *rxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	FlexCAN peripheral base pointer.
mbIdx	The FlexCAN Message Buffer index.

rxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Rx Message Buffer is full and has been read successfully.
kStatus_FLEXCAN_Rx-Overflow	- Rx Message Buffer is already overflowed and has been read successfully.
kStatus_Fail	- Rx Message Buffer is empty.

14.2.7.26 status_t FLEXCAN_TransferReceiveFifoBlocking (CAN_Type ∗ *base,* flexcan_frame_t ∗ *rxFrame*)

Note that a transfer handle does not need to be created before calling this API.

Parameters

base	FlexCAN peripheral base pointer.
rxFrame	Pointer to CAN message frame structure for reception.

Return values

kStatus_Success	- Read Message from Rx FIFO successfully.
kStatus_Fail	- Rx FIFO is not enabled.

14.2.7.27 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

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
callback	The callback function.
userData	The parameter of the callback function.

14.2.7.28 status_t FLEXCAN_TransferSendNonBlocking (CAN_Type * *base,* flexcan_handle_t * *handle,* flexcan_mb_transfer_t * *xfer*)

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

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
xfer	FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

kStatus_Success	Start Tx Message Buffer sending process successfully.
kStatus_Fail	Write Tx Message Buffer failed.
kStatus_FLEXCAN_TxBusy	Tx Message Buffer is in use.

14.2.7.29 status_t FLEXCAN_TransferReceiveNonBlocking (CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_mb_transfer_t * *xfer*)

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

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
xfer	FlexCAN Message Buffer transfer structure. See the flexcan_mb_transfer_t.

Return values

kStatus_Success	- Start Rx Message Buffer receiving process successfully.
kStatus_FLEXCAN_RxBusy	- Rx Message Buffer is in use.

14.2.7.30 status_t FLEXCAN_TransferReceiveFifoNonBlocking (CAN_Type * *base*, flexcan_handle_t * *handle*, flexcan_fifo_transfer_t * *xfer*)

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

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
xfer	FlexCAN Rx FIFO transfer structure. See the flexcan_fifo_transfer_t.

Return values

kStatus_Success	- Start Rx FIFO receiving process successfully.
kStatus_FLEXCAN_RxFifoBusy	- Rx FIFO is currently in use.

14.2.7.31 void FLEXCAN_TransferAbortSend (CAN_Type * *base*, flexcan_handle_t * *handle*, uint8_t *mbIdx*)

This function aborts the interrupt driven message send process.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
mbIdx	The FlexCAN Message Buffer index.

14.2.7.32 void FLEXCAN_TransferAbortReceive (CAN_Type * *base*, flexcan_handle_t * *handle*, uint8_t *mbIdx*)

This function aborts the interrupt driven message receive process.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.
mbIdx	The FlexCAN Message Buffer index.

14.2.7.33 void FLEXCAN_TransferAbortReceiveFifo (CAN_Type * *base*, flexcan_handle_t * *handle*)

This function aborts the interrupt driven message receive from Rx FIFO process.

Parameters

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.

14.2.7.34 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

base	FlexCAN peripheral base address.
handle	FlexCAN handle pointer.

14.3 FlexCAN eDMA Driver

14.3.1 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.

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.
- status_t FLEXCAN_TransferReceiveFifoEDMA (CAN_Type ∗base, flexcan_edma_handle_t ∗handle, flexcan_fifo_transfer_t ∗xfer)
  Receives the CAN Message from the Rx FIFO using eDMA.
- void FLEXCAN_TransferAbortReceiveFifoEDMA (CAN_Type ∗base, flexcan_edma_handle_t ∗handle)
  Aborts the receive process which used eDMA.

14.3.2 Data Structure Documentation

14.3.2.1 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 Rx FIFO channel used.
- volatile uint8_t rxFifoState
  Rx FIFO transfer state.

14.3.2.1.0.49 Field Documentation

14.3.2.1.0.49.1 flexcan_edma_transfer_callback_t flexcan_edma_handle_t::callback

14.3.2.1.0.49.2 void∗ flexcan_edma_handle_t::userData

14.3.2.1.0.49.3 edma_handle_t∗ flexcan_edma_handle_t::rxFifoEdmaHandle

14.3.2.1.0.49.4 volatile uint8_t flexcan_edma_handle_t::rxFifoState

14.3.3 Typedef Documentation

14.3.3.1 typedef void(∗ flexcan_edma_transfer_callback_t)(CAN_Type ∗base, flexcan_edma_handle_t ∗handle, status_t status, void ∗userData)

14.3.4 Function Documentation

14.3.4.1 void FLEXCAN_TransferCreateHandleEDMA (CAN_Type ∗ *base,* flexcan_edma_handle_t ∗ *handle,* flexcan_edma_transfer_callback_t *callback,* void ∗ *userData,* edma_handle_t ∗ *rxFifoEdmaHandle*)

Parameters

base	FlexCAN peripheral base address.
handle	Pointer to flexcan_edma_handle_t structure.
callback	The callback function.
userData	The parameter of the callback function.
rxFifoEdma-Handle	User-requested DMA handle for Rx FIFO DMA transfer.

14.3.4.2 status_t FLEXCAN_TransferReceiveFifoEDMA (CAN_Type * *base*, flexcan_edma_handle_t * *handle*, flexcan_fifo_transfer_t * *xfer*)

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

base	FlexCAN peripheral base address.
handle	Pointer to flexcan_edma_handle_t structure.
xfer	FlexCAN Rx FIFO EDMA transfer structure, see flexcan_fifo_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_FLEXCAN_Rx-FifoBusy	Previous transfer ongoing.

14.3.4.3 void FLEXCAN_TransferAbortReceiveFifoEDMA (CAN_Type * *base*, flexcan_edma_handle_t * *handle*)

This function aborts the receive process which used eDMA.

Parameters

base	FlexCAN peripheral base address.

handle	Pointer to flexcan_edma_handle_t structure.

Chapter 15
GPIO: General-Purpose Input/Output Driver

15.1 Overview

Modules

- FGPIO Driver
- GPIO Driver

Data Structures

- struct gpio_pin_config_t
  The GPIO pin configuration structure. More...

Enumerations

- enum gpio_pin_direction_t {
  kGPIO_DigitalInput = 0U,
  kGPIO_DigitalOutput = 1U }
  GPIO direction definition.

Driver version

- #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))
  GPIO driver version 2.2.0.

15.2 Data Structure Documentation

15.2.1 struct gpio_pin_config_t

Each pin can only be configured as either an output pin or an input pin at a time. If configured as an input pin, leave the outputConfig unused. Note that in some use cases, the corresponding port property should be configured in advance with the PORT_SetPinConfig().

Data Fields

- gpio_pin_direction_t pinDirection
  GPIO direction, input or output.
- uint8_t outputLogic
  Set a default output logic, which has no use in input.

15.3 Macro Definition Documentation

15.3.1 #define FSL_GPIO_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))

15.4 Enumeration Type Documentation

15.4.1 enum gpio_pin_direction_t

Enumerator

kGPIO_DigitalInput Set current pin as digital input.
kGPIO_DigitalOutput Set current pin as digital output.

15.5 GPIO Driver

15.5.1 Overview

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

15.5.2 Typical use case

15.5.2.1 Output Operation

```
/* Output pin configuration */
gpio_pin_config_t led_config =
{
    kGpioDigitalOutput,
    1,
};
/*  Sets the configuration */
GPIO_PinInit(GPIO_LED, LED_PINNUM, &led_config);
```

15.5.2.2 Input Operation

```
/*  Input pin configuration */
PORT_SetPinInterruptConfig(BOARD_SW2_PORT, BOARD_SW2_GPIO_PIN,
      kPORT_InterruptFallingEdge);
NVIC_EnableIRQ(BOARD_SW2_IRQ);
gpio_pin_config_t sw1_config =
{
    kGpioDigitalInput,
    0,
};
/* Sets the input pin configuration */
GPIO_PinInit(GPIO_SW1, SW1_PINNUM, &sw1_config);
```

GPIO Configuration

- void GPIO_PinInit (GPIO_Type *base, uint32_t pin, const gpio_pin_config_t *config)
  Initializes a GPIO pin used by the board.

GPIO Output Operations

- static void GPIO_WritePinOutput (GPIO_Type *base, uint32_t pin, uint8_t output)
  Sets the output level of the multiple GPIO pins to the logic 1 or 0.
- static void GPIO_SetPinsOutput (GPIO_Type *base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 1.
- static void GPIO_ClearPinsOutput (GPIO_Type *base, uint32_t mask)
  Sets the output level of the multiple GPIO pins to the logic 0.
- static void GPIO_TogglePinsOutput (GPIO_Type *base, uint32_t mask)
  Reverses the current output logic of the multiple GPIO pins.

GPIO Input Operations

- static uint32_t GPIO_ReadPinInput (GPIO_Type ∗base, uint32_t pin)
  Reads the current input value of the GPIO port.

GPIO Interrupt

- uint32_t GPIO_GetPinsInterruptFlags (GPIO_Type ∗base)
  Reads the GPIO port interrupt status flag.
- void GPIO_ClearPinsInterruptFlags (GPIO_Type ∗base, uint32_t mask)
  Clears multiple GPIO pin interrupt status flags.

15.5.3 Function Documentation

15.5.3.1 void GPIO_PinInit (GPIO_Type ∗ *base,* uint32_t *pin,* const gpio_pin_config_t ∗ *config*)

To initialize the GPIO, define a pin configuration, as either input or output, in the user file. Then, call the GPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration.

```
* // Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* //Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO port pin number
config	GPIO pin configuration pointer

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO pin number
output	GPIO pin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

15.5.3.4 static void GPIO_ClearPinsOutput (GPIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

15.5.3.5 static void GPIO_TogglePinsOutput (GPIO_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

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

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
pin	GPIO pin number

Return values

GPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

15.5.3.7 uint32_t GPIO_GetPinsInterruptFlags (GPIO_Type ∗ *base*)

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)

Return values

The	current GPIO port interrupt status flag, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

15.5.3.8 void GPIO_ClearPinsInterruptFlags (GPIO_Type ∗ *base,* uint32_t *mask*)

Parameters

base	GPIO peripheral base pointer (GPIOA, GPIOB, GPIOC, and so on.)
mask	GPIO pin number macro

15.6 FGPIO Driver

15.6.1 Overview

This chapter describes the programming interface of the FGPIO driver. The FGPIO driver configures the FGPIO module and provides a functional interface to build the GPIO application.

Note

FGPIO (Fast GPIO) is only available in a few MCUs. FGPIO and GPIO share the same peripheral but use different registers. FGPIO is closer to the core than the regular GPIO and it's faster to read and write.

15.6.2 Typical use case

15.6.2.1 Output Operation

```
/* Output pin configuration */
gpio_pin_config_t led_config =
{
    kGpioDigitalOutput,
    1,
};
/*  Sets the configuration */
FGPIO_PinInit(FGPIO_LED, LED_PINNUM, &led_config);
```

15.6.2.2 Input Operation

```
/*  Input pin configuration */
PORT_SetPinInterruptConfig(BOARD_SW2_PORT, BOARD_SW2_FGPIO_PIN,
      kPORT_InterruptFallingEdge);
NVIC_EnableIRQ(BOARD_SW2_IRQ);
gpio_pin_config_t sw1_config =
{
    kGpioDigitalInput,
    0,
};
/* Sets the input pin configuration */
FGPIO_PinInit(FGPIO_SW1, SW1_PINNUM, &sw1_config);
```

FGPIO Configuration

- void FGPIO_Init (FGPIO_Type *base)
  Initializes the FGPIO peripheral.
- void FGPIO_PinInit (FGPIO_Type *base, uint32_t pin, const gpio_pin_config_t *config)
  Initializes a FGPIO pin used by the board.

FGPIO Output Operations

- static void FGPIO_WritePinOutput (FGPIO_Type *base, uint32_t pin, uint8_t output)

Sets the output level of the multiple FGPIO pins to the logic 1 or 0.
- static void FGPIO_SetPinsOutput (FGPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple FGPIO pins to the logic 1.
- static void FGPIO_ClearPinsOutput (FGPIO_Type ∗base, uint32_t mask)
  Sets the output level of the multiple FGPIO pins to the logic 0.
- static void FGPIO_TogglePinsOutput (FGPIO_Type ∗base, uint32_t mask)
  Reverses the current output logic of the multiple FGPIO pins.

FGPIO Input Operations

- static uint32_t FGPIO_ReadPinInput (FGPIO_Type ∗base, uint32_t pin)
  Reads the current input value of the FGPIO port.

FGPIO Interrupt

- uint32_t FGPIO_GetPinsInterruptFlags (FGPIO_Type ∗base)
  Reads the FGPIO port interrupt status flag.
- void FGPIO_ClearPinsInterruptFlags (FGPIO_Type ∗base, uint32_t mask)
  Clears the multiple FGPIO pin interrupt status flag.

15.6.3 Function Documentation

15.6.3.1 void FGPIO_Init (FGPIO_Type ∗ *base*)

This function ungates the FGPIO clock.

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)

15.6.3.2 void FGPIO_PinInit (FGPIO_Type ∗ *base,* uint32_t *pin,* const gpio_pin_config_t ∗ *config*)

To initialize the FGPIO driver, define a pin configuration, as either input or output, in the user file. Then, call the FGPIO_PinInit() function.

This is an example to define an input pin or an output pin configuration:

```
* // Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
*   kGPIO_DigitalInput,
*   0,
* }
* //Define a digital output pin configuration,
* gpio_pin_config_t config =
```

```
* {
*   kGPIO_DigitalOutput,
*   0,
* }
*
```

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
pin	FGPIO port pin number
config	FGPIO pin configuration pointer

15.6.3.3 static void FGPIO_WritePinOutput (FGPIO_Type ∗ *base,* uint32_t *pin,* uint8_t *output*) [inline], [static]

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
pin	FGPIO pin number
output	FGPIOpin output logic level. • 0: corresponding pin output low-logic level. • 1: corresponding pin output high-logic level.

15.6.3.4 static void FGPIO_SetPinsOutput (FGPIO_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

15.6.3.5 static void FGPIO_ClearPinsOutput (FGPIO_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

15.6.3.6 static void FGPIO_TogglePinsOutput (FGPIO_Type ∗ *base,* uint32_t *mask*) [inline], [static]

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

15.6.3.7 static uint32_t FGPIO_ReadPinInput (FGPIO_Type ∗ *base,* uint32_t *pin*) [inline], [static]

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
pin	FGPIO pin number

Return values

FGPIO	port input value • 0: corresponding pin input low-logic level. • 1: corresponding pin input high-logic level.

15.6.3.8 uint32_t FGPIO_GetPinsInterruptFlags (FGPIO_Type ∗ *base*)

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level-sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)

Return values

The	current FGPIO port interrupt status flags, for example, 0x00010001 means the pin 0 and 17 have the interrupt.

15.6.3.9 void FGPIO_ClearPinsInterruptFlags (FGPIO_Type * *base,* uint32_t *mask*)

Parameters

base	FGPIO peripheral base pointer (FGPIOA, FGPIOB, FGPIOC, and so on.)
mask	FGPIO pin number macro

Chapter 16
I2C: Inter-Integrated Circuit Driver

16.1 Overview

Modules

- I2C DMA Driver
- I2C Driver
- I2C FreeRTOS Driver
- I2C eDMA Driver

16.2 I2C Driver

16.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs target the low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires knowing the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs target the high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions I2C_MasterTransferNonBlocking() set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

16.2.2 Typical use case

16.2.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Gets the default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Inititializes the I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Sends a start and a slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
      kI2C_Write/kI2C_Read);

/* Waits for the sent out address. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{

}

if(status & kI2C_ReceiveNakFlag)
{
    return kStatus_I2C_Nak;
}

result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE,
      kI2C_TransferDefaultFlag);

if(result)
```

```
{
    return result;
}
```

16.2.2.2 Master Operation in interrupt transactional method

```
i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle, status_t status, void *
      userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        g_MasterCompletionFlag = true;
    }
}

/* Gets a default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Initializes the I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
      i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
      masterXfer);

/*  Waits for a transfer to be completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

16.2.2.3 Master Operation in DMA transactional method

```
i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *
      userData)
{
    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
```

```
    {
        g_MasterCompletionFlag = true;
    }
}

/* Gets the default configuration for the master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Initializes the I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMAMGR_RequestChannel((dma_request_source_t)DMA_REQUEST_SRC, 0, &dmaHandle);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &
      g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

/*  Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

16.2.2.4 Slave Operation in functional method

```
i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*A default configuration 7-bit
       addressing mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/
      kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig, I2C_SLAVE_CLK);

/* Waits for an address match. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag))
{

}

/* A slave transmits; master is reading from the slave. */
if (status & kI2C_TransferDirectionFlag)
{
    result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, txBuff, BUFFER_SIZE);
}
else
{
    I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, rxBuff, BUFFER_SIZE);
}

return result;
```

16.2.2.5 Slave Operation in interrupt transactional method

```
i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
      userData)
{
    switch (xfer->event)
    {
        /*  Transmit request */
        case kI2C_SlaveTransmitEvent:
            /*  Update information for transmit process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Receives request */
        case kI2C_SlaveReceiveEvent:
            /*  Update information for received process */
            xfer->data = g_slave_buff;
            xfer->dataSize = I2C_DATA_LENGTH;
            break;

        /*  Transfer is done */
        case kI2C_SlaveCompletionEvent:
            g_SlaveCompletionFlag = true;
            break;

        default:
            g_SlaveCompletionFlag = true;
            break;
    }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*A default configuration 7-bit
       addressing mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/
      kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig, I2C_SLAVE_CLK);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
      kI2C_SlaveCompletionEvent);

/*  Waits for a transfer to be completed. */
while (!g_SlaveCompletionFlag)
{
}
g_SlaveCompletionFlag = false;
```

Data Structures

- struct i2c_master_config_t
  I2C master user configuration. More...
- struct i2c_slave_config_t
  I2C slave user configuration. More...
- struct i2c_master_transfer_t

I2C master transfer structure. More...

- struct i2c_master_handle_t
  I2C master handle structure. More...
- struct i2c_slave_transfer_t
  I2C slave transfer structure. More...
- struct i2c_slave_handle_t
  I2C slave handle structure. More...

Typedefs

- typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t status, void ∗userData)
  I2C master transfer callback typedef.
- typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, i2c_slave_transfer_t ∗xfer, void ∗userData)
  I2C slave transfer callback typedef.

Enumerations

- enum _i2c_status {
  kStatus_I2C_Busy = MAKE_STATUS(kStatusGroup_I2C, 0),
  kStatus_I2C_Idle = MAKE_STATUS(kStatusGroup_I2C, 1),
  kStatus_I2C_Nak = MAKE_STATUS(kStatusGroup_I2C, 2),
  kStatus_I2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_I2C, 3),
  kStatus_I2C_Timeout = MAKE_STATUS(kStatusGroup_I2C, 4),
  kStatus_I2C_Addr_Nak = MAKE_STATUS(kStatusGroup_I2C, 5) }
  I2C status return codes.
- enum _i2c_flags {
  kI2C_ReceiveNakFlag = I2C_S_RXAK_MASK,
  kI2C_IntPendingFlag = I2C_S_IICIF_MASK,
  kI2C_TransferDirectionFlag = I2C_S_SRW_MASK,
  kI2C_RangeAddressMatchFlag = I2C_S_RAM_MASK,
  kI2C_ArbitrationLostFlag = I2C_S_ARBL_MASK,
  kI2C_BusBusyFlag = I2C_S_BUSY_MASK,
  kI2C_AddressMatchFlag = I2C_S_IAAS_MASK,
  kI2C_TransferCompleteFlag = I2C_S_TCF_MASK,
  kI2C_StopDetectFlag = I2C_FLT_STOPF_MASK << 8,
  kI2C_StartDetectFlag = I2C_FLT_STARTF_MASK << 8 }
  I2C peripheral flags.
- enum _i2c_interrupt_enable {
  kI2C_GlobalInterruptEnable = I2C_C1_IICIE_MASK,
  kI2C_StartStopDetectInterruptEnable = I2C_FLT_SSIE_MASK }
  I2C feature interrupt source.
- enum i2c_direction_t {
  kI2C_Write = 0x0U,

kI2C_Read = 0x1U }
The direction of master and slave transfers.

- enum i2c_slave_address_mode_t {
  kI2C_Address7bit = 0x0U,
  kI2C_RangeMatch = 0X2U }
  Addressing mode.
- enum _i2c_master_transfer_flags {
  kI2C_TransferDefaultFlag = 0x0U,
  kI2C_TransferNoStartFlag = 0x1U,
  kI2C_TransferRepeatedStartFlag = 0x2U,
  kI2C_TransferNoStopFlag = 0x4U }
  I2C transfer control flag.
- enum i2c_slave_transfer_event_t {
  kI2C_SlaveAddressMatchEvent = 0x01U,
  kI2C_SlaveTransmitEvent = 0x02U,
  kI2C_SlaveReceiveEvent = 0x04U,
  kI2C_SlaveTransmitAckEvent = 0x08U,
  kI2C_SlaveStartEvent = 0x10U,
  kI2C_SlaveCompletionEvent = 0x20U,
  kI2C_SlaveGenaralcallEvent = 0x40U,
  kI2C_SlaveAllEvents }
  Set of events sent to the callback for nonblocking slave transfers.

Driver version

- #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  I2C driver version 2.0.3.

Initialization and deinitialization

- void I2C_MasterInit (I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes the I2C peripheral.
- void I2C_SlaveInit (I2C_Type ∗base, const i2c_slave_config_t ∗slaveConfig, uint32_t srcClock_Hz)
  Initializes the I2C peripheral.
- void I2C_MasterDeinit (I2C_Type ∗base)
  De-initializes the I2C master peripheral.
- void I2C_SlaveDeinit (I2C_Type ∗base)
  De-initializes the I2C slave peripheral.
- void I2C_MasterGetDefaultConfig (i2c_master_config_t ∗masterConfig)
  Sets the I2C master configuration structure to default values.
- void I2C_SlaveGetDefaultConfig (i2c_slave_config_t ∗slaveConfig)
  Sets the I2C slave configuration structure to default values.
- static void I2C_Enable (I2C_Type ∗base, bool enable)
  Enables or disabless the I2C peripheral operation.

Status

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

Interrupts

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

DMA Control

- static void I2C_EnableDMA (I2C_Type ∗base, bool enable)
  Enables/disables the I2C DMA interrupt.
- static uint32_t I2C_GetDataRegAddr (I2C_Type ∗base)
  Gets the I2C tx/rx data register address.

Bus Operations

- void I2C_MasterSetBaudRate (I2C_Type ∗base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the I2C master transfer baud rate.
- status_t I2C_MasterStart (I2C_Type ∗base, uint8_t address, i2c_direction_t direction)
  Sends a START on the I2C bus.
- status_t I2C_MasterStop (I2C_Type ∗base)
  Sends a STOP signal on the I2C bus.
- status_t I2C_MasterRepeatedStart (I2C_Type ∗base, uint8_t address, i2c_direction_t direction)
  Sends a REPEATED START on the I2C bus.
- status_t I2C_MasterWriteBlocking (I2C_Type ∗base, const uint8_t ∗txBuff, size_t txSize, uint32_t flags)
  Performs a polling send transaction on the I2C bus.
- status_t I2C_MasterReadBlocking (I2C_Type ∗base, uint8_t ∗rxBuff, size_t rxSize, uint32_t flags)
  Performs a polling receive transaction on the I2C bus.
- status_t I2C_SlaveWriteBlocking (I2C_Type ∗base, const uint8_t ∗txBuff, size_t txSize)
  Performs a polling send transaction on the I2C bus.
- void I2C_SlaveReadBlocking (I2C_Type ∗base, uint8_t ∗rxBuff, size_t rxSize)
  Performs a polling receive transaction on the I2C bus.
- status_t I2C_MasterTransferBlocking (I2C_Type ∗base, i2c_master_transfer_t ∗xfer)
  Performs a master polling transfer on the I2C bus.

Transactional

- void I2C_MasterTransferCreateHandle (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *userData)
  Initializes the I2C handle which is used in transactional functions.
- status_t I2C_MasterTransferNonBlocking (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)
  Performs a master interrupt non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCount (I2C_Type *base, i2c_master_handle_t *handle, size_t *count)
  Gets the master transfer status during a interrupt non-blocking transfer.
- void I2C_MasterTransferAbort (I2C_Type *base, i2c_master_handle_t *handle)
  Aborts an interrupt non-blocking transfer early.
- void I2C_MasterTransferHandleIRQ (I2C_Type *base, void *i2cHandle)
  Master interrupt handler.
- void I2C_SlaveTransferCreateHandle (I2C_Type *base, i2c_slave_handle_t *handle, i2c_slave_transfer_callback_t callback, void *userData)
  Initializes the I2C handle which is used in transactional functions.
- status_t I2C_SlaveTransferNonBlocking (I2C_Type *base, i2c_slave_handle_t *handle, uint32_t eventMask)
  Starts accepting slave transfers.
- void I2C_SlaveTransferAbort (I2C_Type *base, i2c_slave_handle_t *handle)
  Aborts the slave transfer.
- status_t I2C_SlaveTransferGetCount (I2C_Type *base, i2c_slave_handle_t *handle, size_t *count)
  Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.
- void I2C_SlaveTransferHandleIRQ (I2C_Type *base, void *i2cHandle)
  Slave interrupt handler.

16.2.3 Data Structure Documentation

16.2.3.1 struct i2c_master_config_t

Data Fields

- bool enableMaster
  Enables the I2C peripheral at initialization time.
- bool enableStopHold
  Controls the stop hold enable.
- bool enableDoubleBuffering
  Controls double buffer enable; notice that enabling the double buffer disables the clock stretch.
- uint32_t baudRate_Bps
  Baud rate configuration of I2C peripheral.
- uint8_t glitchFilterWidth
  Controls the width of the glitch.

16.2.3.1.0.50 Field Documentation

16.2.3.1.0.50.1 bool i2c_master_config_t::enableMaster

16.2.3.1.0.50.2 bool i2c_master_config_t::enableStopHold

16.2.3.1.0.50.3 bool i2c_master_config_t::enableDoubleBuffering

16.2.3.1.0.50.4 uint32_t i2c_master_config_t::baudRate_Bps

16.2.3.1.0.50.5 uint8_t i2c_master_config_t::glitchFilterWidth

16.2.3.2 struct i2c_slave_config_t

Data Fields

- bool enableSlave
  Enables the I2C peripheral at initialization time.
- bool enableGeneralCall
  Enables the general call addressing mode.
- bool enableWakeUp
  Enables/disables waking up MCU from low-power mode.
- bool enableDoubleBuffering
  Controls a double buffer enable; notice that enabling the double buffer disables the clock stretch.
- bool enableBaudRateCtl
  Enables/disables independent slave baud rate on SCL in very fast I2C modes.
- uint16_t slaveAddress
  A slave address configuration.
- uint16_t upperAddress
  A maximum boundary slave address used in a range matching mode.
- i2c_slave_address_mode_t addressingMode
  An addressing mode configuration of i2c_slave_address_mode_config_t.
- uint32_t sclStopHoldTime_ns
  the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition), SDA hold time and SCL start hold time are also configured according to the SCL stop hold time.

16.2.3.2.0.51 Field Documentation

16.2.3.2.0.51.1 bool i2c_slave_config_t::enableSlave

16.2.3.2.0.51.2 bool i2c_slave_config_t::enableGeneralCall

16.2.3.2.0.51.3 bool i2c_slave_config_t::enableWakeUp

16.2.3.2.0.51.4 bool i2c_slave_config_t::enableDoubleBuffering

16.2.3.2.0.51.5 bool i2c_slave_config_t::enableBaudRateCtl

16.2.3.2.0.51.6 uint16_t i2c_slave_config_t::slaveAddress

16.2.3.2.0.51.7 uint16_t i2c_slave_config_t::upperAddress

16.2.3.2.0.51.8 i2c_slave_address_mode_t i2c_slave_config_t::addressingMode

16.2.3.2.0.51.9 uint32_t i2c_slave_config_t::sclStopHoldTime_ns

16.2.3.3 struct i2c_master_transfer_t

Data Fields

- uint32_t flags
  A transfer flag which controls the transfer.
- uint8_t slaveAddress
  7-bit slave address.
- i2c_direction_t direction
  A transfer direction, read or write.
- uint32_t subaddress
  A sub address.
- uint8_t subaddressSize
  A size of the command buffer.
- uint8_t *volatile data
  A transfer buffer.
- volatile size_t dataSize
  A transfer size.

16.2.3.3.0.52 Field Documentation

16.2.3.3.0.52.1 uint32_t i2c_master_transfer_t::flags

16.2.3.3.0.52.2 uint8_t i2c_master_transfer_t::slaveAddress

16.2.3.3.0.52.3 i2c_direction_t i2c_master_transfer_t::direction

16.2.3.3.0.52.4 uint32_t i2c_master_transfer_t::subaddress

Transferred MSB first.

16.2.3.3.0.52.5 uint8_t i2c_master_transfer_t::subaddressSize

16.2.3.3.0.52.6 uint8_t∗ volatile i2c_master_transfer_t::data

16.2.3.3.0.52.7 volatile size_t i2c_master_transfer_t::dataSize

16.2.3.4 struct _i2c_master_handle

I2C master handle typedef.

Data Fields

- i2c_master_transfer_t transfer
  I2C master transfer copy.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t state
  A transfer state maintained during transfer.
- i2c_master_transfer_callback_t completionCallback
  A callback function called when the transfer is finished.
- void ∗ userData
  A callback parameter passed to the callback function.

16.2.3.4.0.53 Field Documentation

16.2.3.4.0.53.1 i2c_master_transfer_t i2c_master_handle_t::transfer

16.2.3.4.0.53.2 size_t i2c_master_handle_t::transferSize

16.2.3.4.0.53.3 uint8_t i2c_master_handle_t::state

16.2.3.4.0.53.4 i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback

16.2.3.4.0.53.5 void∗ i2c_master_handle_t::userData

16.2.3.5 struct i2c_slave_transfer_t

Data Fields

- i2c_slave_transfer_event_t event
  A reason that the callback is invoked.
- uint8_t ∗volatile data
  A transfer buffer.
- volatile size_t dataSize
  A transfer size.
- status_t completionStatus
  Success or error code describing how the transfer completed.
- size_t transferredCount
  A number of bytes actually transferred since the start or since the last repeated start.

16.2.3.5.0.54 Field Documentation

16.2.3.5.0.54.1 i2c_slave_transfer_event_t i2c_slave_transfer_t::event

16.2.3.5.0.54.2 uint8_t* volatile i2c_slave_transfer_t::data

16.2.3.5.0.54.3 volatile size_t i2c_slave_transfer_t::dataSize

16.2.3.5.0.54.4 status_t i2c_slave_transfer_t::completionStatus

Only applies for kI2C_SlaveCompletionEvent.

16.2.3.5.0.54.5 size_t i2c_slave_transfer_t::transferredCount

16.2.3.6 struct _i2c_slave_handle

I2C slave handle typedef.

Data Fields

- volatile bool isBusy
  Indicates whether a transfer is busy.
- i2c_slave_transfer_t transfer
  I2C slave transfer copy.
- uint32_t eventMask
  A mask of enabled events.
- i2c_slave_transfer_callback_t callback
  A callback function called at the transfer event.
- void * userData
  A callback parameter passed to the callback.

16.2.3.6.0.55 Field Documentation

16.2.3.6.0.55.1 volatile bool i2c_slave_handle_t::isBusy

16.2.3.6.0.55.2 i2c_slave_transfer_t i2c_slave_handle_t::transfer

16.2.3.6.0.55.3 uint32_t i2c_slave_handle_t::eventMask

16.2.3.6.0.55.4 i2c_slave_transfer_callback_t i2c_slave_handle_t::callback

16.2.3.6.0.55.5 void∗ i2c_slave_handle_t::userData

16.2.4 Macro Definition Documentation

16.2.4.1 #define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

16.2.5 Typedef Documentation

16.2.5.1 typedef void(∗ i2c_master_transfer_callback_t)(I2C_Type ∗base, i2c_master_handle_t ∗handle, status_t status, void ∗userData)

16.2.5.2 typedef void(∗ i2c_slave_transfer_callback_t)(I2C_Type ∗base, i2c_slave_transfer_t ∗xfer, void ∗userData)

16.2.6 Enumeration Type Documentation

16.2.6.1 enum _i2c_status

Enumerator

kStatus_I2C_Busy I2C is busy with current transfer.
kStatus_I2C_Idle Bus is Idle.
kStatus_I2C_Nak NAK received during transfer.
kStatus_I2C_ArbitrationLost Arbitration lost during transfer.
kStatus_I2C_Timeout Wait event timeout.
kStatus_I2C_Addr_Nak NAK received during the address probe.

16.2.6.2 enum _i2c_flags

The following status register flags can be cleared:

- kI2C_ArbitrationLostFlag
- kI2C_IntPendingFlag
- kI2C_StartDetectFlag
- kI2C_StopDetectFlag

Note

These enumerations are meant to be OR'd together to form a bit mask.

Enumerator

kI2C_ReceiveNakFlag I2C receive NAK flag.
kI2C_IntPendingFlag I2C interrupt pending flag.
kI2C_TransferDirectionFlag I2C transfer direction flag.
kI2C_RangeAddressMatchFlag I2C range address match flag.
kI2C_ArbitrationLostFlag I2C arbitration lost flag.
kI2C_BusBusyFlag I2C bus busy flag.
kI2C_AddressMatchFlag I2C address match flag.
kI2C_TransferCompleteFlag I2C transfer complete flag.
kI2C_StopDetectFlag I2C stop detect flag.
kI2C_StartDetectFlag I2C start detect flag.

16.2.6.3 enum _i2c_interrupt_enable

Enumerator

kI2C_GlobalInterruptEnable I2C global interrupt.
kI2C_StartStopDetectInterruptEnable I2C start&stop detect interrupt.

16.2.6.4 enum i2c_direction_t

Enumerator

kI2C_Write Master transmits to the slave.
kI2C_Read Master receives from the slave.

16.2.6.5 enum i2c_slave_address_mode_t

Enumerator

kI2C_Address7bit 7-bit addressing mode.
kI2C_RangeMatch Range address match addressing mode.

16.2.6.6 enum _i2c_master_transfer_flags

Enumerator

kI2C_TransferDefaultFlag A transfer starts with a start signal, stops with a stop signal.

kI2C_TransferNoStartFlag A transfer starts without a start signal.
kI2C_TransferRepeatedStartFlag A transfer starts with a repeated start signal.
kI2C_TransferNoStopFlag A transfer ends without a stop signal.

16.2.6.7 enum i2c_slave_transfer_event_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to I2C_SlaveTransferNonBlocking() to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

kI2C_SlaveAddressMatchEvent Received the slave address after a start or repeated start.
kI2C_SlaveTransmitEvent A callback is requested to provide data to transmit (slave-transmitter role).
kI2C_SlaveReceiveEvent A callback is requested to provide a buffer in which to place received data (slave-receiver role).
kI2C_SlaveTransmitAckEvent A callback needs to either transmit an ACK or NACK.
kI2C_SlaveStartEvent A start/repeated start was detected.
kI2C_SlaveCompletionEvent A stop was detected or finished transfer, completing the transfer.
kI2C_SlaveGenaralcallEvent Received the general call address after a start or repeated start.
kI2C_SlaveAllEvents A bit mask of all available events.

16.2.7 Function Documentation

16.2.7.1 void I2C_MasterInit (I2C_Type * *base*, const i2c_master_config_t * *masterConfig*, uint32_t *srcClock_Hz*)

Call this API to ungate the I2C clock and configure the I2C with master configuration.

Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can be custom filled or it can be set with default values by using the I2C_MasterGetDefaultConfig(). After calling this API, the master is ready to transfer. This is an example.

```
* i2c_master_config_t config = {
* .enableMaster = true,
* .enableStopHold = false,
* .highDrive = false,
* .baudRate_Bps = 100000,
```

```
* .glitchFilterWidth = 0
* };
* I2C_MasterInit(I2C0, &config, 12000000U);
*
```

Parameters

base	I2C base pointer
masterConfig	A pointer to the master configuration structure
srcClock_Hz	I2C peripheral clock frequency in Hz

16.2.7.2 void I2C_SlaveInit (I2C_Type * *base*, const i2c_slave_config_t * *slaveConfig*, uint32_t *srcClock_Hz*)

Call this API to ungate the I2C clock and initialize the I2C with the slave configuration.

Note

This API should be called at the beginning of the application. Otherwise, any operation to the I2C module can cause a hard fault because the clock is not enabled. The configuration structure can partly be set with default values by I2C_SlaveGetDefaultConfig() or it can be custom filled by the user. This is an example.

```
* i2c_slave_config_t config = {
* .enableSlave = true,
* .enableGeneralCall = false,
* .addressingMode = kI2C_Address7bit,
* .slaveAddress = 0x1DU,
* .enableWakeUp = false,
* .enablehighDrive = false,
* .enableBaudRateCtl = false,
* .sclStopHoldTime_ns = 4000
* };
* I2C_SlaveInit(I2C0, &config, 12000000U);
*
```

Parameters

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

16.2.7.3 void I2C_MasterDeinit (I2C_Type * *base*)

Call this API to gate the I2C clock. The I2C master module can't work unless the I2C_MasterInit is called.

Parameters

base	I2C base pointer

16.2.7.4 void I2C_SlaveDeinit (I2C_Type * *base*)

Calling this API gates the I2C clock. The I2C slave module can't work unless the I2C_SlaveInit is called to enable the clock.

Parameters

base	I2C base pointer

16.2.7.5 void I2C_MasterGetDefaultConfig (i2c_master_config_t * *masterConfig*)

The purpose of this API is to get the configuration structure initialized for use in the I2C_MasterConfigure(). Use the initialized structure unchanged in the I2C_MasterConfigure() or modify the structure before calling the I2C_MasterConfigure(). This is an example.

```
* i2c_master_config_t config;
* I2C_MasterGetDefaultConfig(&config);
*
```

Parameters

masterConfig	A pointer to the master configuration structure.

16.2.7.6 void I2C_SlaveGetDefaultConfig (i2c_slave_config_t * *slaveConfig*)

The purpose of this API is to get the configuration structure initialized for use in the I2C_SlaveConfigure(). Modify fields of the structure before calling the I2C_SlaveConfigure(). This is an example.

```
* i2c_slave_config_t config;
* I2C_SlaveGetDefaultConfig(&config);
*
```

Parameters

slaveConfig	A pointer to the slave configuration structure.

16.2.7.7 static void I2C_Enable (I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	I2C base pointer
enable	Pass true to enable and false to disable the module.

16.2.7.8 uint32_t I2C_MasterGetStatusFlags (I2C_Type * *base*)

Parameters

base	I2C base pointer

Returns

status flag, use status flag to AND _i2c_flags to get the related status.

16.2.7.9 static uint32_t I2C_SlaveGetStatusFlags (I2C_Type * *base*) [inline], [static]

Parameters

base	I2C base pointer

Returns

status flag, use status flag to AND _i2c_flags to get the related status.

16.2.7.10 static void I2C_MasterClearStatusFlags (I2C_Type * *base*, uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag.

Parameters

base	I2C base pointer
statusMask	The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: • kI2C_StartDetectFlag (if available) • kI2C_StopDetectFlag (if available) • kI2C_ArbitrationLostFlag • kI2C_IntPendingFlagFlag

16.2.7.11 static void I2C_SlaveClearStatusFlags (I2C_Type ∗ *base,* uint32_t *statusMask*) [inline], [static]

The following status register flags can be cleared kI2C_ArbitrationLostFlag and kI2C_IntPendingFlag

Parameters

base	I2C base pointer
statusMask	The status flag mask, defined in type i2c_status_flag_t. The parameter can be any combination of the following values: • kI2C_StartDetectFlag (if available) • kI2C_StopDetectFlag (if available) • kI2C_ArbitrationLostFlag • kI2C_IntPendingFlagFlag

16.2.7.12 void I2C_EnableInterrupts (I2C_Type ∗ *base,* uint32_t *mask*)

Parameters

base	I2C base pointer
mask	interrupt source The parameter can be combination of the following source if defined: • kI2C_GlobalInterruptEnable • kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable • kI2C_SdaTimeoutInterruptEnable

16.2.7.13 void I2C_DisableInterrupts (I2C_Type ∗ *base,* uint32_t *mask*)

Parameters

base	I2C base pointer
mask	interrupt source The parameter can be combination of the following source if defined: • kI2C_GlobalInterruptEnable • kI2C_StopDetectInterruptEnable/kI2C_StartDetectInterruptEnable • kI2C_SdaTimeoutInterruptEnable

16.2.7.14 static void I2C_EnableDMA (I2C_Type * *base*, bool *enable*) [inline], [static]

Parameters

base	I2C base pointer
enable	true to enable, false to disable

16.2.7.15 static uint32_t I2C_GetDataRegAddr (I2C_Type * *base*) [inline], [static]

This API is used to provide a transfer address for I2C DMA transfer configuration.

Parameters

base	I2C base pointer

Returns

data register address

16.2.7.16 void I2C_MasterSetBaudRate (I2C_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

Parameters

base	I2C base pointer
baudRate_Bps	the baud rate value in bps
srcClock_Hz	Source clock

16.2.7.17 status_t I2C_MasterStart (I2C_Type * *base,* uint8_t *address,* i2c_direction_t *direction*)

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

base	I2C peripheral base pointer
address	7-bit slave device address.
direction	Master transfer directions(transmit/receive).

Return values

kStatus_Success	Successfully send the start signal.
kStatus_I2C_Busy	Current bus is busy.

16.2.7.18 status_t I2C_MasterStop (I2C_Type ∗ *base*)

Return values

kStatus_Success	Successfully send the stop signal.
kStatus_I2C_Timeout	Send stop signal failed, timeout.

16.2.7.19 status_t I2C_MasterRepeatedStart (I2C_Type ∗ *base,* uint8_t *address,* i2c_direction_t *direction*)

Parameters

base	I2C peripheral base pointer
address	7-bit slave device address.
direction	Master transfer directions(transmit/receive).

Return values

kStatus_Success	Successfully send the start signal.
kStatus_I2C_Busy	Current bus is busy but not occupied by current I2C master.

16.2.7.20 status_t I2C_MasterWriteBlocking (I2C_Type ∗ *base,* const uint8_t ∗ *txBuff,* size_t *txSize,* uint32_t *flags*)

Parameters

base	The I2C peripheral base pointer.
txBuff	The pointer to the data to be transferred.
txSize	The length in bytes of the data to be transferred.
flags	Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_ArbitrationLost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

16.2.7.21 status_t I2C_MasterReadBlocking (I2C_Type * *base*, uint8_t * *rxBuff*, size_t *rxSize*, uint32_t *flags*)

Note

The I2C_MasterReadBlocking function stops the bus before reading the final byte. Without stopping the bus prior for the final read, the bus issues another read, resulting in garbage data being read into the data register.

Parameters

base	I2C peripheral base pointer.
rxBuff	The pointer to the data to store the received data.
rxSize	The length in bytes of the data to be received.
flags	Transfer control flag to decide whether need to send a stop, use kI2C_TransferDefaultFlag to issue a stop and kI2C_TransferNoStop to not send a stop.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Timeout	Send stop signal failed, timeout.

16.2.7.22 status_t I2C_SlaveWriteBlocking (I2C_Type * *base*, const uint8_t * *txBuff*, size_t *txSize*)

Parameters

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

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

16.2.7.23 void I2C_SlaveReadBlocking (I2C_Type ∗ *base,* uint8_t ∗ *rxBuff,* size_t *rxSize*)

Parameters

base	I2C peripheral base pointer.
rxBuff	The pointer to the data to store the received data.
rxSize	The length in bytes of the data to be received.

16.2.7.24 status_t I2C_MasterTransferBlocking (I2C_Type ∗ *base,* i2c_master_transfer_t ∗ *xfer*)

Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

base	I2C peripheral base address.
xfer	Pointer to the transfer structure.

Return values

kStatus_Success	Successfully complete the data transmission.
kStatus_I2C_Busy	Previous transmission still not finished.
kStatus_I2C_Timeout	Transfer error, wait signal timeout.
kStatus_I2C_Arbitration-Lost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

16.2.7.25 void I2C_MasterTransferCreateHandle (I2C_Type ∗ *base,* i2c_master_handle_t ∗ *handle,* i2c_master_transfer_callback_t *callback,* void ∗ *userData*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure to store the transfer state.
callback	pointer to user callback function.
userData	user parameter passed to the callback function.

16.2.7.26 status_t I2C_MasterTransferNonBlocking (I2C_Type ∗ *base,* i2c_master_handle_t ∗ *handle,* i2c_master_transfer_t ∗ *xfer*)

Note

Calling the API returns immediately after transfer initiates. The user needs to call I2C_MasterGetTransferCount to poll the transfer status to check whether the transfer is finished. If the return status is not kStatus_I2C_Busy, the transfer is finished.

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure which stores the transfer state.
xfer	pointer to i2c_master_transfer_t structure.

Return values

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

16.2.7.27 status_t I2C_MasterTransferGetCount (I2C_Type * *base,* i2c_master_handle_t * *handle,* size_t * *count*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure which stores the transfer state.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

16.2.7.28 void I2C_MasterTransferAbort (I2C_Type * *base*, i2c_master_handle_t * *handle*)

Note

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

Parameters

base	I2C base pointer.
handle	pointer to i2c_master_handle_t structure which stores the transfer state

16.2.7.29 void I2C_MasterTransferHandleIRQ (I2C_Type * *base*, void * *i2cHandle*)

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_master_handle_t structure.

16.2.7.30 void I2C_SlaveTransferCreateHandle (I2C_Type * *base*, i2c_slave_handle_t * *handle*, i2c_slave_transfer_callback_t *callback*, void * *userData*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure to store the transfer state.
callback	pointer to user callback function.
userData	user parameter passed to the callback function.

16.2.7.31 status_t I2C_SlaveTransferNonBlocking (I2C_Type * *base*, i2c_slave_handle_t * *handle*, uint32_t *eventMask*)

Call this API after calling the I2C_SlaveInit() and I2C_SlaveTransferCreateHandle() to start processing transactions driven by an I2C master. The slave monitors the I2C bus and passes events to the callback that was passed into the call to I2C_SlaveTransferCreateHandle(). The callback is always invoked from the interrupt context.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of i2c_slave_transfer_event_t enumerators for the events you wish to receive. The kI2C_SlaveTransmitEvent and #kLPI2C_SlaveReceiveEvent events are always enabled and do not need to be included in the mask. Alternatively, pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the kI2C_SlaveAllEvents constant is provided as a convenient way to enable all events.

Parameters

base	The I2C peripheral base address.
handle	Pointer to #i2c_slave_handle_t structure which stores the transfer state.
eventMask	Bit mask formed by OR'ing together i2c_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 kI2C_SlaveAllEvents to enable all events.

Return values

#kStatus_Success	Slave transfers were successfully started.
kStatus_I2C_Busy	Slave transfers have already been started on this handle.

16.2.7.32 void I2C_SlaveTransferAbort (I2C_Type * *base*, i2c_slave_handle_t * *handle*)

Note

This API can be called at any time to stop slave for handling the bus events.

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure which stores the transfer state.

16.2.7.33 status_t I2C_SlaveTransferGetCount (I2C_Type * *base,* i2c_slave_handle_t * *handle,* size_t * *count*)

Parameters

base	I2C base pointer.
handle	pointer to i2c_slave_handle_t structure.
count	Number of bytes transferred so far by the non-blocking transaction.

Return values

kStatus_InvalidArgument	count is Invalid.
kStatus_Success	Successfully return the count.

16.2.7.34 void I2C_SlaveTransferHandleIRQ (I2C_Type * *base,* void * *i2cHandle*)

Parameters

base	I2C base pointer.
i2cHandle	pointer to i2c_slave_handle_t structure which stores the transfer state

16.3 I2C eDMA Driver

16.3.1 Overview

Data Structures

- struct i2c_master_edma_handle_t
  I2C master eDMA transfer structure. More...

Typedefs

- typedef void(∗ i2c_master_edma_transfer_callback_t)(I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, status_t status, void ∗userData)
  I2C master eDMA transfer callback typedef.

I2C Block eDMA Transfer Operation

- void I2C_MasterCreateEDMAHandle (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, i2c_master_edma_transfer_callback_t callback, void ∗userData, edma_handle_t ∗edmaHandle)
  Initializes the I2C handle which is used in transcational functions.
- status_t I2C_MasterTransferEDMA (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, i2c_master_transfer_t ∗xfer)
  Performs a master eDMA non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCountEDMA (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, size_t ∗count)
  Gets a master transfer status during the eDMA non-blocking transfer.
- void I2C_MasterTransferAbortEDMA (I2C_Type ∗base, i2c_master_edma_handle_t ∗handle)
  Aborts a master eDMA non-blocking transfer early.

16.3.2 Data Structure Documentation

16.3.2.1 struct _i2c_master_edma_handle

I2C master eDMA handle typedef.

Data Fields

- i2c_master_transfer_t transfer
  I2C master transfer structure.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t nbytes
  eDMA minor byte transfer count initially configured.
- uint8_t state

I2C master transfer status.

- edma_handle_t ∗ dmaHandle
  The eDMA handler used.
- i2c_master_edma_transfer_callback_t completionCallback
  A callback function called after the eDMA transfer is finished.
- void ∗ userData
  A callback parameter passed to the callback function.

16.3.2.1.0.56 Field Documentation

16.3.2.1.0.56.1 i2c_master_transfer_t i2c_master_edma_handle_t::transfer

16.3.2.1.0.56.2 size_t i2c_master_edma_handle_t::transferSize

16.3.2.1.0.56.3 uint8_t i2c_master_edma_handle_t::nbytes

16.3.2.1.0.56.4 uint8_t i2c_master_edma_handle_t::state

16.3.2.1.0.56.5 edma_handle_t∗ i2c_master_edma_handle_t::dmaHandle

16.3.2.1.0.56.6 i2c_master_edma_transfer_callback_t i2c_master_edma_handle_t::completion-Callback

16.3.2.1.0.56.7 void∗ i2c_master_edma_handle_t::userData

16.3.3 Typedef Documentation

16.3.3.1 typedef void(∗ i2c_master_edma_transfer_callback_t)(I2C_Type ∗base, i2c_master_edma_handle_t ∗handle, status_t status, void ∗userData)

16.3.4 Function Documentation

16.3.4.1 void I2C_MasterCreateEDMAHandle (I2C_Type ∗ *base,* i2c_master_edma_-handle_t ∗ *handle,* i2c_master_edma_transfer_callback_t *callback,* void ∗ *userData,* edma_handle_t ∗ *edmaHandle*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.
callback	A pointer to the user callback function.

userData	A user parameter passed to the callback function.
edmaHandle	eDMA handle pointer.

16.3.4.2 status_t I2C_MasterTransferEDMA (I2C_Type * *base*, i2c_master_edma_handle_t * *handle*, i2c_master_transfer_t * *xfer*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.
xfer	A pointer to the transfer structure of i2c_master_transfer_t.

Return values

kStatus_Success	Sucessfully completed the data transmission.
kStatus_I2C_Busy	A previous transmission is still not finished.
kStatus_I2C_Timeout	Transfer error, waits for a signal timeout.
kStatus_I2C_ArbitrationLost	Transfer error, arbitration lost.
kStataus_I2C_Nak	Transfer error, receive NAK during transfer.

16.3.4.3 status_t I2C_MasterTransferGetCountEDMA (I2C_Type * *base*, i2c_master_edma_handle_t * *handle*, size_t * *count*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.
count	A number of bytes transferred by the non-blocking transaction.

16.3.4.4 void I2C_MasterTransferAbortEDMA (I2C_Type * *base*, i2c_master_edma_handle_t * *handle*)

Parameters

base	I2C peripheral base address.
handle	A pointer to the i2c_master_edma_handle_t structure.

16.4 I2C DMA Driver

16.4.1 Overview

Data Structures

- struct i2c_master_dma_handle_t
  I2C master DMA transfer structure. More...

Typedefs

- typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)
  I2C master DMA transfer callback typedef.

I2C Block DMA Transfer Operation

- void I2C_MasterTransferCreateHandleDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, i2c_master_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗dmaHandle)
  Initializes the I2C handle which is used in transcational functions.
- status_t I2C_MasterTransferDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, i2c_-master_transfer_t ∗xfer)
  Performs a master DMA non-blocking transfer on the I2C bus.
- status_t I2C_MasterTransferGetCountDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, size_t ∗count)
  Gets a master transfer status during a DMA non-blocking transfer.
- void I2C_MasterTransferAbortDMA (I2C_Type ∗base, i2c_master_dma_handle_t ∗handle)
  Aborts a master DMA non-blocking transfer early.

16.4.2 Data Structure Documentation

16.4.2.1 struct _i2c_master_dma_handle

I2C master DMA handle typedef.

Data Fields

- i2c_master_transfer_t transfer
  I2C master transfer struct.
- size_t transferSize
  Total bytes to be transferred.
- uint8_t state
  I2C master transfer status.
- dma_handle_t ∗ dmaHandle

The DMA handler used.

- i2c_master_dma_transfer_callback_t completionCallback
  A callback function called after the DMA transfer finished.
- void ∗ userData
  A callback parameter passed to the callback function.

16.4.2.1.0.57 Field Documentation

16.4.2.1.0.57.1 i2c_master_transfer_t i2c_master_dma_handle_t::transfer

16.4.2.1.0.57.2 size_t i2c_master_dma_handle_t::transferSize

16.4.2.1.0.57.3 uint8_t i2c_master_dma_handle_t::state

16.4.2.1.0.57.4 dma_handle_t∗ i2c_master_dma_handle_t::dmaHandle

16.4.2.1.0.57.5 i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completion-Callback

16.4.2.1.0.57.6 void∗ i2c_master_dma_handle_t::userData

16.4.3 Typedef Documentation

16.4.3.1 typedef void(∗ i2c_master_dma_transfer_callback_t)(I2C_Type ∗base, i2c_master_dma_handle_t ∗handle, status_t status, void ∗userData)

16.4.4 Function Documentation

16.4.4.1 void I2C_MasterTransferCreateHandleDMA (I2C_Type ∗ *base,* i2c_master_dma_handle_t ∗ *handle,* i2c_master_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *dmaHandle*)

Parameters

base	I2C peripheral base address
handle	Pointer to the i2c_master_dma_handle_t structure
callback	Pointer to the user callback function
userData	A user parameter passed to the callback function
dmaHandle	DMA handle pointer

16.4.4.2 status_t I2C_MasterTransferDMA (I2C_Type ∗ *base,* i2c_master_dma_handle_t ∗ *handle,* i2c_master_transfer_t ∗ *xfer*)

Parameters

base	I2C peripheral base address
handle	A pointer to the i2c_master_dma_handle_t structure
xfer	A pointer to the transfer structure of the i2c_master_transfer_t

Return values

kStatus_Success	Sucessfully completes the data transmission.
kStatus_I2C_Busy	A previous transmission is still not finished.
kStatus_I2C_Timeout	A transfer error, waits for the signal timeout.
kStatus_I2C_Arbitration-Lost	A transfer error, arbitration lost.
kStataus_I2C_Nak	A transfer error, receives NAK during transfer.

16.4.4.3 status_t I2C_MasterTransferGetCountDMA (I2C_Type ∗ *base,* i2c_master_dma_handle_t ∗ *handle,* size_t ∗ *count*)

Parameters

base	I2C peripheral base address
handle	A pointer to the i2c_master_dma_handle_t structure
count	A number of bytes transferred so far by the non-blocking transaction.

16.4.4.4 void I2C_MasterTransferAbortDMA (I2C_Type ∗ *base,* i2c_master_dma_handle_t ∗ *handle*)

Parameters

base	I2C peripheral base address
handle	A pointer to the i2c_master_dma_handle_t structure.

16.5 I2C FreeRTOS Driver

16.5.1 Overview

I2C RTOS Operation

- status_t I2C_RTOS_Init (i2c_rtos_handle_t ∗handle, I2C_Type ∗base, const i2c_master_config_t ∗masterConfig, uint32_t srcClock_Hz)
  Initializes I2C.
- status_t I2C_RTOS_Deinit (i2c_rtos_handle_t ∗handle)
  Deinitializes the I2C.
- status_t I2C_RTOS_Transfer (i2c_rtos_handle_t ∗handle, i2c_master_transfer_t ∗transfer)
  Performs the I2C transfer.

16.5.2 Function Documentation

16.5.2.1 status_t I2C_RTOS_Init (i2c_rtos_handle_t ∗ *handle,* I2C_Type ∗ *base,* const i2c_master_config_t ∗ *masterConfig,* uint32_t *srcClock_Hz*)

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

Parameters

handle	The RTOS I2C handle, the pointer to an allocated space for RTOS context.
base	The pointer base address of the I2C instance to initialize.
masterConfig	The configuration structure to set-up I2C in master mode.
srcClock_Hz	The frequency of an input clock of the I2C module.

Returns

status of the operation.

16.5.2.2 status_t I2C_RTOS_Deinit (i2c_rtos_handle_t ∗ *handle*)

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

Parameters

handle	The RTOS I2C handle.

16.5.2.3 status_t I2C_RTOS_Transfer (i2c_rtos_handle_t * *handle,* i2c_master_transfer_t * *transfer*)

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

Parameters

handle	The RTOS I2C handle.
transfer	A structure specifying the transfer parameters.

Returns

status of the operation.

Chapter 17
LIN: Local Interconnect Network Driver

17.1 Overview

Modules

- LIN Driver
- LIN LPUART Driver

17.2 LIN Driver

17.2.1 Overview

The KSDK provides a peripheral driver for the Local Interconnect Network (LIN) module of Kinetis devices.

17.2.2 Typical use case

17.2.2.1 LIN Operation

For example of Local Interconnect Network please see driver example lin_master or lin_slave.

Data Structures

- struct lin_user_config_t
  LIN hardware configuration structure. More...
- struct lin_state_t
  Runtime state of the LIN driver. More...
- struct lin_baudrate_values_t
  Structure of baudrate properties. More...
- struct lin_frame_struct
  Informations of frame. More...
- struct lin_protocol_user_config_t
  Protocol configuration structure. More...
- struct lin_product_id
  Product id structure. More...
- struct lin_serial_number
  Serial number. More...
- struct lin_protocol_state_t
  Protocol state structure. More...
- struct lin_node_attribute
  Attributes of LIN node. More...
- struct lin_master_data_t
  LIN master data structure. More...

Typedefs

- typedef void(∗ lin_timer_get_time_interval_t)(uint32_t ∗nanoSeconds)
  Callback function to get time interval in nano seconds.
- typedef void(∗ lin_callback_t)(uint32_t instance, void ∗linState)
  LIN Driver callback function type.

Enumerations

- enum lin_frame_type {
  LIN_FRM_UNCD = 0x00U,
  LIN_FRM_EVNT = 0x01U,
  LIN_FRM_SPRDC = 0x10U,
  LIN_FRM_DIAG = 0x11U }
  List of LIN frame types.
- enum lin_frame_response {
  LIN_RES_NOTHING = 0x00U,
  LIN_RES_PUB = 0x01U,
  LIN_RES_SUB = 0x10U }
  List of LIN frame response types.
- enum lin_diagnostic_mode {
  DIAG_NONE = 0x00U,
  DIAG_INTERLEAVE_MODE = 0x01U,
  DIAG_ONLY_MODE = 0x02U }
  Diagnostic mode.
- enum lin_event_id_t {
  LIN_NO_EVENT = 0x00U,
  LIN_WAKEUP_SIGNAL = 0x01U,
  LIN_BAUDRATE_ADJUSTED = 0x02U,
  LIN_RECV_BREAK_FIELD_OK = 0x03U,
  LIN_RECV_SYNC_OK = 0x04U,
  LIN_RECV_SYNC_ERROR = 0x05U,
  LIN_PID_OK = 0x06U,
  LIN_PID_ERROR = 0x07U,
  LIN_FRAME_ERROR = 0x08U,
  LIN_READBACK_ERROR = 0x09U,
  LIN_CHECKSUM_ERROR = 0x0AU,
  LIN_TX_COMPLETED = 0x0BU,
  LIN_RX_COMPLETED = 0x0CU,
  LIN_NO_DATA_TIMEOUT = 0x0DU,
  LIN_BUS_ACTIVITY_TIMEOUT = 0x0EU,
  LIN_TIMEOUT_ERROR = 0x0FU }
  Defines types for an enumerating event related to an Identifier.
- enum lin_status_t {
  LIN_IFC_NOT_SUPPORT = 0x00U,
  LIN_INITIALIZED = 0x01U,
  LIN_SUCCESS = 0x02U,
  LIN_ERROR = 0x03U,
  LIN_TX_BUSY = 0x04U,
  LIN_RX_BUSY = 0x05U,
  LIN_BUS_BUSY = 0x06U,
  LIN_NO_TRANSFER_IN_PROGRESS = 0x07U,
  LIN_TIMEOUT = 0x08U,

LIN_LPUART_STAT_CLOCK_GATED_OFF = 0x09U }
Defines Error codes of the LIN driver.

- enum lin_node_state_t {
  LIN_NODE_STATE_UNINIT = 0x00U,
  LIN_NODE_STATE_SLEEP_MODE = 0x01U,
  LIN_NODE_STATE_IDLE = 0x02U,
  LIN_NODE_STATE_SEND_BREAK_FIELD = 0x03U,
  LIN_NODE_STATE_RECV_SYNC = 0x04U,
  LIN_NODE_STATE_SEND_PID = 0x05U,
  LIN_NODE_STATE_RECV_PID = 0x06U,
  LIN_NODE_STATE_RECV_DATA = 0x07U,
  LIN_NODE_STATE_RECV_DATA_COMPLETED = 0x08U,
  LIN_NODE_STATE_SEND_DATA = 0x09U,
  LIN_NODE_STATE_SEND_DATA_COMPLETED = 0x0AU }
  Define type for an enumerating LIN Node state.
- enum lin_protocol_handle {
  LIN_PROTOCOL_21 = 0x00U,
  LIN_PROTOCOL_J2602 = 0x01U }
  List of protocols.
- enum lin_supported_baudrates_t
  List of supported baudrates for autobaud feature.

Variables

- uint8_t lin_user_config_t::hardware_instance
  interface instance number
- uint32_t lin_user_config_t::baudRate
  baudrate of LIN Hardware Interface to configure
- bool lin_user_config_t::nodeFunction
  Node function as Master or Slave.
- bool lin_user_config_t::autobaudEnable
  Enable Autobaud feature.
- lin_timer_get_time_interval_t lin_user_config_t::timerGetTimeIntervalCallback
  Callback function to get time interval in nano seconds.
- const uint8_t * lin_state_t::txBuff
  The buffer of data being sent.
- uint8_t * lin_state_t::rxBuff
  The buffer of received data.
- uint8_t lin_state_t::cntByte
  To count number of bytes already transmitted or received.
- volatile uint8_t lin_state_t::txSize
  The remaining number of bytes to be received.
- volatile uint8_t lin_state_t::rxSize
  The remaining number of bytes to be received.
- uint8_t lin_state_t::checkSum
  Checksum byte.
- volatile bool lin_state_t::isTxBusy
  True if the LIN interface is transmitting frame data.

- volatile bool lin_state_t::isRxBusy
  True if the LIN interface is receiving frame data.
- volatile bool lin_state_t::isBusBusy
  True if there are data, frame headers being transferred on bus.
- volatile bool lin_state_t::isTxBlocking
  True if transmit is blocking transaction.
- volatile bool lin_state_t::isRxBlocking
  True if receive is blocking transaction.
- lin_callback_t lin_state_t::Callback
  Callback function to invoke after receiving a byte or transmitting a byte.
- uint8_t lin_state_t::currentId
  Current ID.
- uint8_t lin_state_t::currentPid
  Current PID.
- volatile lin_event_id_t lin_state_t::currentEventId
  Current ID Event.
- volatile lin_node_state_t lin_state_t::currentNodeState
  Current Node state.
- volatile uint32_t lin_state_t::timeoutCounter
  Value of the timeout counter.
- volatile bool lin_state_t::timeoutCounterFlag
  Timeout counter flag.
- volatile bool lin_state_t::baudrateEvalEnable
  Baudrate Evaluation Process Enable.
- volatile uint8_t lin_state_t::fallingEdgeInterruptCount
  Falling Edge count of a sync byte.
- uint32_t lin_state_t::linSourceClockFreq
  Frequency of the source clock for LIN.
- volatile bool lin_state_t::txCompleted
  Used to wait for LIN interface ISR to complete transmission.
- volatile bool lin_state_t::rxCompleted
  Used to wait for LIN interface ISR to complete reception.
- lin_frame_type lin_frame_struct::frm_type
  Frame information (unconditional or event triggered..)
- uint8_t lin_frame_struct::frm_len
  Length of the frame.
- lin_frame_response lin_frame_struct::frm_response
  Action response when received PID.
- uint8_t lin_frame_struct::frm_offset
  Frame byte offset in frame buffer.
- uint8_t lin_frame_struct::flag_offset
  Flag byte offset in flag buffer.
- uint8_t lin_frame_struct::flag_size
  Flag size in flag buffer.
- uint32_t lin_frame_struct::delay
  Frame delay.
- const uint8_t * lin_frame_struct::frame_data_ptr
  List of Signal to which the frame is associated and its offset.
- lin_protocol_handle lin_protocol_user_config_t::protocol_version
  Protocol version.
- lin_protocol_handle lin_protocol_user_config_t::language_version

Language version.

- uint8_t lin_protocol_user_config_t::number_of_configurable_frames
  Number of frame except diagnostic frames.
- uint8_t lin_protocol_user_config_t::frame_start
  Start index of frame list.
- const lin_frame_struct * lin_protocol_user_config_t::frame_tbl_ptr
  Frame list except diagnostic frames.
- const uint16_t * lin_protocol_user_config_t::list_identifiers_ROM_ptr
  Configuration in ROM.
- uint8_t * lin_protocol_user_config_t::list_identifiers_RAM_ptr
  Configuration in RAM.
- uint16_t lin_protocol_user_config_t::max_idle_timeout_cnt
  Max Idle timeout counter.
- uint16_t lin_protocol_user_config_t::max_message_length
  Max message length.
- uint16_t lin_product_id::supplier_id
  Supplier ID.
- uint16_t lin_product_id::function_id
  Function ID.
- uint8_t lin_product_id::variant
  Variant value.
- uint8_t lin_serial_number::serial_0
  Serial 0.
- uint8_t lin_serial_number::serial_1
  Serial 1.
- uint8_t lin_serial_number::serial_2
  Serial 2.
- uint8_t lin_serial_number::serial_3
  Serial 3.
- uint16_t lin_protocol_state_t::baud_rate
  Adjusted baud rate.
- uint8_t * lin_protocol_state_t::response_buffer_ptr
  Response buffer.
- uint8_t lin_protocol_state_t::response_length
  Response length.
- uint8_t lin_protocol_state_t::successful_transfer
  Transfer flag.
- uint8_t lin_protocol_state_t::error_in_response
  Error response.
- uint8_t lin_protocol_state_t::timeout_in_response
  Error response.
- bool lin_protocol_state_t::go_to_sleep_flg
  Go to sleep flag.
- uint8_t lin_protocol_state_t::current_id
  Current PID.
- uint8_t lin_protocol_state_t::num_of_processed_frame
  Number of processed frames.
- uint8_t lin_protocol_state_t::num_of_successfull_frame
  Number of processed frames.
- uint8_t lin_protocol_state_t::next_transmit_tick
  Used to count the next transmit tick.

- bool lin_protocol_state_t::save_config_flg
  Set when save configuration request has been received.
- lin_diagnostic_mode lin_protocol_state_t::diagnostic_mode
  Diagnostic mode.
- uint16_t lin_protocol_state_t::frame_timeout_cnt
  Frame timeout counter.
- uint16_t lin_protocol_state_t::idle_timeout_cnt
  Idle timeout counter.
- bool lin_protocol_state_t::transmit_error_resp_sig_flg
  Flag indicates that the error reponse signal is going to be sent.
- bool lin_protocol_state_t::event_trigger_collision_flg
  Flag indicates collision on bus.
- uint8_t ∗ lin_node_attribute::configured_NAD_ptr
  NAD value used in configuration command.
- uint8_t lin_node_attribute::initial_NAD
  Initial NAD.
- lin_product_id lin_node_attribute::product_id
  Product ID.
- lin_serial_number lin_node_attribute::serial_number
  Serial number.
- uint8_t ∗ lin_node_attribute::resp_err_frm_id_ptr
  List index of frame contain response error signal.
- uint8_t lin_node_attribute::num_frame_have_esignal
  Number of frame contain error signal.
- uint8_t lin_node_attribute::response_error_byte_offset
  Byte offset of response error signal.
- uint8_t lin_node_attribute::response_error_bit_offset
  Bit offset of response error signal.
- uint8_t lin_node_attribute::num_of_fault_state_signal
  Number of Fault state signal.
- uint16_t lin_node_attribute::P2_min
  P2 min.
- uint16_t lin_node_attribute::ST_min
  ST min.
- uint16_t lin_node_attribute::N_As_timeout
  N_As_timeout.
- uint16_t lin_node_attribute::N_Cr_timeout
  N_Cr_timeout.
- uint8_t lin_master_data_t::active_schedule_id
  Active schedule table id.
- uint8_t lin_master_data_t::previous_schedule_id
  Previous schedule table id.
- uint8_t ∗ lin_master_data_t::schedule_start_entry_ptr
  Start entry of each schedule table.
- uint8_t lin_master_data_t::data_buffer [8]
  Master data buffer.
- LPUART_Type ∗const g_linLpuartBase [FSL_FEATURE_SOC_LPUART_COUNT]
  Table of base addresses for LPUART instances.
- const IRQn_Type g_linLpuartRxTxIrqId [FSL_FEATURE_SOC_LPUART_COUNT]
  Table to save LPUART IRQ enumeration numbers defined in the CMSIS header file.
- lin_state_t ∗ g_linStatePtr [FSL_FEATURE_SOC_LPUART_COUNT]

Table to save LPUART state structure pointers.

- lin_user_config_t * g_linUserconfigPtr [FSL_FEATURE_SOC_LPUART_COUNT]
  Table to save LIN user config structure pointers.

LIN DRIVER

- uint32_t LIN_CalcMaxHeaderTimeoutCnt (uint32_t baudRate)
  Calculates maximal header time lenght.
- uint32_t LIN_CalcMaxResTimeoutCnt (uint32_t baudRate, uint8_t size)
  Calculates maximal header time lenght.
- lin_status_t LIN_SetResponse (uint8_t instance, uint8_t *response_buff, uint8_t response_length, uint8_t max_frame_res_timeout)
  Forwards a response to a lower level.
- lin_status_t LIN_RxResponse (uint8_t instance, uint8_t *response_buff, uint8_t response_length, uint8_t max_frame_res_timeout)
  Forwards a response to a higher level.
- lin_status_t LIN_IgnoreResponse (uint8_t instance)
  Put a node into idle state.
- void LIN_GetSlaveDefaultConfig (lin_user_config_t *linUserConfig)
  Initializes linUserConfig variable for a slave node.
- void LIN_GetMasterDefaultConfig (lin_user_config_t *linUserConfig)
  Initializes linUserConfig variable for a master node.
- void LIN_CalculateBaudrate (uint32_t instance, uint32_t baudRate_Bps, uint32_t srcClock_Hz, uint32_t *osr, uint16_t *sbr)
  Calculates baudrate registers values for given baudrate.
- void LIN_SetBaudrate (uint32_t instance, uint32_t osr, uint16_t sbr)
  Set baudrate registers values.
- lin_status_t LIN_Init (uint32_t instance, lin_user_config_t *linUserConfig, lin_state_t *linCurrent-State, uint32_t clockSource)
  Initializes an instance LIN Hardware Interface for LIN Network.
- lin_status_t LIN_Deinit (uint32_t instance)
  Shuts down the LIN Hardware Interface by disabling interrupts and transmitter/receiver.
- lin_callback_t LIN_InstallCallback (uint32_t instance, lin_callback_t function)
  Installs callback function that is used for LIN_DRV_IRQHandler.
- lin_status_t LIN_SendFrameDataBlocking (uint32_t instance, const uint8_t *txBuff, uint8_t txSize, uint32_t timeoutMSec)
  Sends Frame data out through the LIN Hardware Interface using blocking method.
- lin_status_t LIN_SendFrameData (uint32_t instance, const uint8_t *txBuff, uint8_t txSize)
  Sends frame data out through the LIN Hardware Interface using non-blocking method.
- lin_status_t LIN_GetTransmitStatus (uint32_t instance, uint8_t *bytesRemaining)
  Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission.
- lin_status_t LIN_ReceiveFrameDataBlocking (uint32_t instance, uint8_t *rxBuff, uint8_t rxSize, uint32_t timeoutMSec)
  Receives frame data through the LIN Hardware Interface using blocking method.
- lin_status_t LIN_ReceiveFrameData (uint32_t instance, uint8_t *rxBuff, uint8_t rxSize)
  Receives frame data through the LIN Hardware Interface using non- blocking method.
- lin_status_t LIN_AbortTransferData (uint32_t instance)
  Aborts an on-going non-blocking transmission/reception.

- lin_status_t LIN_GetReceiveStatus (uint32_t instance, uint8_t *bytesRemaining)
  Get status of an on-going non-blocking reception While receiving frame data using non-blocking method, users can use this function to get status of that receiving.
- lin_status_t LIN_GoToSleepMode (uint32_t instance)
  Puts current LIN node to sleep mode This function changes current node state to LIN_NODE_STATE_SLEEP_MODE.
- lin_status_t LIN_GotoIdleState (uint32_t instance)
  Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.
- lin_status_t LIN_SendWakeupSignal (uint32_t instance)
  Sends a wakeup signal through the LIN Hardware Interface.
- lin_node_state_t LIN_GetCurrentNodeState (uint32_t instance)
  Get the current LIN node state.
- void LIN_TimeoutService (uint32_t instance)
  Callback function for Timer Interrupt Handler Users may use (optional, not required) LIN_TimeoutService to check if timeout has occured during non-blocking frame data transmission and reception.
- void LIN_SetTimeoutCounter (uint32_t instance, uint32_t timeoutValue)
  Set Value for Timeout Counter that is used in LIN_TimeoutService.
- lin_status_t LIN_MasterSendHeader (uint32_t instance, uint8_t id)
  Sends frame header out through the LIN Hardware Interface using a non-blocking method.
- lin_status_t LIN_EnableIRQ (uint32_t instance)
  Enables LIN hardware interrupts.
- lin_status_t LIN_DisableIRQ (uint32_t instance)
  Disables LIN hardware interrupts.
- void LIN_IRQHandler (uint8_t instance)
  Interrupt handler for LIN Hardware Interface.
- lin_status_t LIN_AutoBaudCapture (uint32_t instance)
  This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful.
- uint8_t LIN_ProcessParity (uint8_t PID, uint8_t typeAction)
  Makes or checks parity bits.
- uint8_t LIN_MakeChecksumByte (const uint8_t *buffer, uint8_t sizeBuffer, uint8_t PID)
  Makes the checksum byte for a frame.

17.2.3 Data Structure Documentation

17.2.3.1 struct lin_user_config_t

Data Fields

- uint8_t hardware_instance
  interface instance number
- uint32_t baudRate
  baudrate of LIN Hardware Interface to configure
- bool nodeFunction
  Node function as Master or Slave.
- bool autobaudEnable
  Enable Autobaud feature.
- lin_timer_get_time_interval_t timerGetTimeIntervalCallback
  Callback function to get time interval in nano seconds.

17.2.3.2 struct lin_state_t

Note that the caller provides memory for the driver state structures during initialization because the driver does not statically allocate memory.

Data Fields

- const uint8_t * txBuff
  The buffer of data being sent.
- uint8_t * rxBuff
  The buffer of received data.
- uint8_t cntByte
  To count number of bytes already transmitted or received.
- volatile uint8_t txSize
  The remaining number of bytes to be received.
- volatile uint8_t rxSize
  The remaining number of bytes to be received.
- uint8_t checkSum
  Checksum byte.
- volatile bool isTxBusy
  True if the LIN interface is transmitting frame data.
- volatile bool isRxBusy
  True if the LIN interface is receiving frame data.
- volatile bool isBusBusy
  True if there are data, frame headers being transferred on bus.
- volatile bool isTxBlocking
  True if transmit is blocking transaction.
- volatile bool isRxBlocking
  True if receive is blocking transaction.
- lin_callback_t Callback
  Callback function to invoke after receiving a byte or transmitting a byte.
- uint8_t currentId
  Current ID.
- uint8_t currentPid
  Current PID.
- volatile lin_event_id_t currentEventId
  Current ID Event.
- volatile lin_node_state_t currentNodeState
  Current Node state.
- volatile uint32_t timeoutCounter
  Value of the timeout counter.
- volatile bool timeoutCounterFlag
  Timeout counter flag.
- volatile bool baudrateEvalEnable
  Baudrate Evaluation Process Enable.
- volatile uint8_t fallingEdgeInterruptCount
  Falling Edge count of a sync byte.
- uint32_t linSourceClockFreq
  Frequency of the source clock for LIN.
- volatile bool txCompleted

Used to wait for LIN interface ISR to complete transmission.
- volatile bool rxCompleted
  Used to wait for LIN interface ISR to complete reception.

17.2.3.3 struct lin_baudrate_values_t

17.2.3.4 struct lin_frame_struct

Data Fields

- lin_frame_type frm_type
  Frame information (unconditional or event triggered..)
- uint8_t frm_len
  Length of the frame.
- lin_frame_response frm_response
  Action response when received PID.
- uint8_t frm_offset
  Frame byte offset in frame buffer.
- uint8_t flag_offset
  Flag byte offset in flag buffer.
- uint8_t flag_size
  Flag size in flag buffer.
- uint32_t delay
  Frame delay.
- const uint8_t * frame_data_ptr
  List of Signal to which the frame is associated and its offset.

17.2.3.5 struct lin_protocol_user_config_t

Data Fields

- lin_protocol_handle protocol_version
  Protocol version.
- lin_protocol_handle language_version
  Language version.
- uint8_t number_of_configurable_frames
  Number of frame except diagnostic frames.
- uint8_t frame_start
  Start index of frame list.
- const lin_frame_struct * frame_tbl_ptr
  Frame list except diagnostic frames.
- const uint16_t * list_identifiers_ROM_ptr
  Configuration in ROM.
- uint8_t * list_identifiers_RAM_ptr
  Configuration in RAM.
- uint16_t max_idle_timeout_cnt
  Max Idle timeout counter.
- uint16_t max_message_length
  Max message length.

17.2.3.6 struct lin_product_id

Data Fields

- uint16_t supplier_id
  Supplier ID.
- uint16_t function_id
  Function ID.
- uint8_t variant
  Variant value.

17.2.3.7 struct lin_serial_number

Data Fields

- uint8_t serial_0
  Serial 0.
- uint8_t serial_1
  Serial 1.
- uint8_t serial_2
  Serial 2.
- uint8_t serial_3
  Serial 3.

17.2.3.8 struct lin_protocol_state_t

Data Fields

- uint16_t baud_rate
  Adjusted baud rate.
- uint8_t * response_buffer_ptr
  Response buffer.
- uint8_t response_length
  Response length.
- uint8_t successful_transfer
  Transfer flag.
- uint8_t error_in_response
  Error response.
- uint8_t timeout_in_response
  Error response.
- bool go_to_sleep_flg
  Go to sleep flag.
- uint8_t current_id
  Current PID.
- uint8_t num_of_processed_frame
  Number of processed frames.
- uint8_t num_of_successfull_frame
  Number of processed frames.
- uint8_t next_transmit_tick

Used to count the next transmit tick.
- bool save_config_flg
  Set when save configuration request has been received.
- lin_diagnostic_mode diagnostic_mode
  Diagnostic mode.
- uint16_t frame_timeout_cnt
  Frame timeout counter.
- uint16_t idle_timeout_cnt
  Idle timeout counter.
- bool transmit_error_resp_sig_flg
  Flag indicates that the error reponse signal is going to be sent.
- bool event_trigger_collision_flg
  Flag indicates collision on bus.

17.2.3.9 struct lin_node_attribute

Data Fields

- uint8_t * configured_NAD_ptr
  NAD value used in configuration command.
- uint8_t initial_NAD
  Initial NAD.
- lin_product_id product_id
  Product ID.
- lin_serial_number serial_number
  Serial number.
- uint8_t * resp_err_frm_id_ptr
  List index of frame contain response error signal.
- uint8_t num_frame_have_esignal
  Number of frame contain error signal.
- uint8_t response_error_byte_offset
  Byte offset of response error signal.
- uint8_t response_error_bit_offset
  Bit offset of response error signal.
- uint8_t num_of_fault_state_signal
  Number of Fault state signal.
- uint16_t P2_min
  P2 min.
- uint16_t ST_min
  ST min.
- uint16_t N_As_timeout
  N_As_timeout.
- uint16_t N_Cr_timeout
  N_Cr_timeout.

17.2.3.10 struct lin_master_data_t

Data Fields

- uint8_t active_schedule_id
  Active schedule table id.
- uint8_t previous_schedule_id
  Previous schedule table id.
- uint8_t * schedule_start_entry_ptr
  Start entry of each schedule table.
- uint8_t data_buffer [8]
  Master data buffer.

17.2.4 Enumeration Type Documentation

17.2.4.1 enum lin_frame_type

Enumerator

LIN_FRM_UNCD unconditional frame
LIN_FRM_EVNT event triggered frame
LIN_FRM_SPRDC sporadic frame
LIN_FRM_DIAG diagnostic frame

17.2.4.2 enum lin_frame_response

Enumerator

LIN_RES_NOTHING node is neither publisher nor subscriber
LIN_RES_PUB node is publisher
LIN_RES_SUB node is subscriber

17.2.4.3 enum lin_diagnostic_mode

Enumerator

DIAG_NONE None.
DIAG_INTERLEAVE_MODE Interleave mode.
DIAG_ONLY_MODE Diagnostic only mode.

17.2.4.4 enum lin_event_id_t

Enumerator

LIN_NO_EVENT No event yet.

LIN_WAKEUP_SIGNAL Received a wakeup signal.
LIN_BAUDRATE_ADJUSTED Indicate that baudrate was adjusted to Master's baudrate.
LIN_RECV_BREAK_FIELD_OK Indicate that correct Break Field was received.
LIN_RECV_SYNC_OK Sync byte is correct.
LIN_RECV_SYNC_ERROR Sync byte is incorrect.
LIN_PID_OK PID correct.
LIN_PID_ERROR PID incorrect.
LIN_FRAME_ERROR Framing Error.
LIN_READBACK_ERROR Readback data is incorrect.
LIN_CHECKSUM_ERROR Checksum byte is incorrect.
LIN_TX_COMPLETED Sending data completed.
LIN_RX_COMPLETED Receiving data completed.
LIN_NO_DATA_TIMEOUT No data timeout.
LIN_BUS_ACTIVITY_TIMEOUT Bus activity timeout.
LIN_TIMEOUT_ERROR Indicate that timeout has occured.

17.2.4.5 enum lin_status_t

Enumerator

LIN_IFC_NOT_SUPPORT This interface is not supported.
LIN_INITIALIZED LIN Hardware has been initialized.
LIN_SUCCESS Successfully done.
LIN_ERROR Error.
LIN_TX_BUSY Transmitter is busy.
LIN_RX_BUSY Receiver is busy.
LIN_BUS_BUSY Bus is busy.
LIN_NO_TRANSFER_IN_PROGRESS No data transfer is in progress.
LIN_TIMEOUT Timeout.
LIN_LPUART_STAT_CLOCK_GATED_OFF LPUART is gated from clock manager.

17.2.4.6 enum lin_node_state_t

Enumerator

LIN_NODE_STATE_UNINIT Uninitialized state.
LIN_NODE_STATE_SLEEP_MODE Sleep mode state.
LIN_NODE_STATE_IDLE Idle state.
LIN_NODE_STATE_SEND_BREAK_FIELD Send break field state.
LIN_NODE_STATE_RECV_SYNC Receive the synchronization byte state.
LIN_NODE_STATE_SEND_PID Send PID state.
LIN_NODE_STATE_RECV_PID Receive PID state.
LIN_NODE_STATE_RECV_DATA Receive data state.
LIN_NODE_STATE_RECV_DATA_COMPLETED Receive data completed state.

LIN_NODE_STATE_SEND_DATA Send data state.
LIN_NODE_STATE_SEND_DATA_COMPLETED Send data completed state.

17.2.4.7 enum lin_protocol_handle

Enumerator

LIN_PROTOCOL_21 LIN protocol version 2.1.
LIN_PROTOCOL_J2602 J2602 protocol.

17.2.5 Function Documentation

17.2.5.1 uint32_t LIN_CalcMaxHeaderTimeoutCnt (uint32_t *baudRate*)

Theader_Maximum = 1.4 ∗ THeader_Nominal, THeader_Nominal = 34 ∗ TBit, (13 nominal bits of breack; 1 nominal bit of break delimiter; 10 bits for SYNC and 10 bits of PID) The function is not include time for conveying break and break delimeter TIME_OUT_UNIT is in micro second

Parameters

baudRate	baudrate

17.2.5.2 uint32_t LIN_CalcMaxResTimeoutCnt (uint32_t *baudRate,* uint8_t *size*)

TResponse_Maximum = 1.4 ∗ TResponse_Nominal, TResponse_Nominal = 10 ∗ (NData+ 1) ∗ TBit

Parameters

baudRate	Baudrate
size	Frame size

17.2.5.3 lin_status_t LIN_SetResponse (uint8_t *instance,* uint8_t ∗ *response_buff,* uint8_t *response_length,* uint8_t *max_frame_res_timeout*)

Parameters

instance	LPUART instance
response_buff	response message
response_-length	length of response
max_frame_-res_timeout	maximal timeout duration for message

Returns

An error code or lin_status_t

17.2.5.4 lin_status_t LIN_RxResponse (uint8_t *instance,* uint8_t ∗ *response_buff,* uint8_t *response_length,* uint8_t *max_frame_res_timeout*)

Parameters

instance	LPUART instance
response_buff	response message
response_-length	length of response
max_frame_-res_timeout	maximal timeout duration for message

Returns

An error code or lin_status_t

17.2.5.5 lin_status_t LIN_IgnoreResponse (uint8_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.6 void LIN_GetSlaveDefaultConfig (lin_user_config_t ∗ *linUserConfig*)

Parameters

linUserConfig	Pointer to LIN user config structure

17.2.5.7 void LIN_GetMasterDefaultConfig (lin_user_config_t ∗ *linUserConfig*)

Parameters

linUserConfig	Pointer to LIN user config structure

17.2.5.8 void LIN_CalculateBaudrate (uint32_t *instance,* uint32_t *baudRate_Bps,* uint32_t *srcClock_Hz,* uint32_t ∗ *osr,* uint16_t ∗ *sbr*)

Parameters

instance	LPUART instance
instance	baudRate_Bps LPUART baudrate
instance	srcClock_Hz LPUART clock frequency
instance	osr LPUART baudrate OSR value, return value
instance	sbr LPUART baudrate SBR value, return value

17.2.5.9 void LIN_SetBaudrate (uint32_t *instance,* uint32_t *osr,* uint16_t *sbr*)

Parameters

instance	LPUART instance
instance	osr LPUART baudrate OSR value
instance	sbr LPUART baudrate SBR value

17.2.5.10 lin_status_t LIN_Init (uint32_t *instance,* lin_user_config_t ∗ *linUserConfig,* lin_state_t ∗ *linCurrentState,* uint32_t *clockSource*)

The caller provides memory for the driver state structures during initialization. The user must select the LIN Hardware Interface clock source in the application to initialize the LIN Hardware Interface.

Parameters

instance	LPUART instance
linUserConfig	user configuration structure of type lin_user_config_t
linCurrentState	pointer to the LIN Hardware Interface driver state structure

Returns

An error code or lin_status_t

17.2.5.11 lin_status_t LIN_Deinit (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.12 lin_callback_t LIN_InstallCallback (uint32_t *instance,* lin_callback_t *function*)

Note

After a callback is installed, it bypasses part of the LIN Hardware Interface IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

Parameters

instance	LPUART instance.
function	the LIN receive callback function.

Returns

Former LIN callback function pointer.

17.2.5.13 lin_status_t LIN_SendFrameDataBlocking (uint32_t *instance,* const uint8_t * *txBuff,* uint8_t *txSize,* uint32_t *timeoutMSec*)

This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

instance	LPUART instance
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send
timeoutMSec	timeout value in milli seconds

Returns

An error code or lin_status_t LIN_BUS_BUSY if the bus is currently busy, transmission cannot be started.

17.2.5.14 lin_status_t LIN_SendFrameData (uint32_t *instance,* const uint8_t ∗ *txBuff,* uint8_t *txSize*)

This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Note

If users use LIN_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN_SetTimeoutCounter(instance, timeoutValue). The timeout value should be big enough to complete the transmission. Timeout in real time is (timeoutValue) ∗ (time period that LIN_TimeoutService is called). For example, if LIN_TimeoutService is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeoutValue ∗ 500 micro seconds.

Parameters

instance	LPUART instance
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send

Returns

An error code or lin_status_t LIN_BUS_BUSY if the bus is currently busy, transmission cannot be started. LIN_SUCCESS if the transmission was completed.

17.2.5.15 lin_status_t LIN_GetTransmitStatus (uint32_t *instance,* uint8_t ∗ *bytesRemaining*)

The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

instance	LPUART instance
bytes-Remaining	Number of bytes still needed to transmit

Returns

lin_status_t LIN_TX_BUSY if the transmission is still in progress. LIN_TIMEOUT if timeout occured and transmission was not completed. LIN_SUCCESS if the transmission was successful.

17.2.5.16 lin_status_t LIN_ReceiveFrameDataBlocking (uint32_t *instance,* uint8_t ∗ *rxBuff,* uint8_t *rxSize,* uint32_t *timeoutMSec*)

This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

instance	LPUART instance
rxBuff	buffer containing 8-bit received data
rxSize	the number of bytes to receive
timeoutMSec	timeout value in milli seconds

Returns

An error code or lin_status_t

17.2.5.17 lin_status_t LIN_ReceiveFrameData (uint32_t *instance,* uint8_t ∗ *rxBuff,* uint8_t *rxSize*)

This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Note

If users use LIN_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN_SetTimeoutCounter(instance, timeoutValue). The timeout value should be big enough to complete the reception. Timeout in real time is (timeoutValue) ∗ (time period that LIN_TimeoutService is called). For example, if LIN_TimeoutService is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeoutValue ∗ 500 micro seconds.

Parameters

instance	LPUART instance
rxBuff	buffer containing 8-bit received data
rxSize	the number of bytes to receive

Returns

An error code or lin_status_t

17.2.5.18 lin_status_t LIN_AbortTransferData (uint32_t *instance*)

While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.19 lin_status_t LIN_GetReceiveStatus (uint32_t *instance,* uint8_t * *bytesRemaining*)

This function return the current event ID, LIN_RX_BUSY while receiving and return LIN_SUCCESS, or timeout (LIN_TIMEOUT) when the reception is complete. The bytesRemaining shows number of bytes that still needed to receive.

Parameters

instance	LPUART instance
bytes-Remaining	Number of bytes still needed to receive

Returns

lin_status_t LIN_RX_BUSY, LIN_TIMEOUT or LIN_SUCCESS

17.2.5.20 lin_status_t LIN_GoToSleepMode (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.21 lin_status_t LIN_GotoIdleState (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.22 lin_status_t LIN_SendWakeupSignal (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.23 lin_node_state_t LIN_GetCurrentNodeState (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

current LIN node state

17.2.5.24 void LIN_TimeoutService (uint32_t *instance*)

User may initialize a timer (for example FTM) in Output Compare Mode with period of 500 micro seconds (recommended). In timer IRQ handler, call this function.

Parameters

instance	LPUART instance

Returns

void

17.2.5.25 void LIN_SetTimeoutCounter (uint32_t *instance,* uint32_t *timeoutValue*)

Parameters

instance	LPUART instance
timeoutValue	Timeout Value to be set

Returns

void

17.2.5.26 lin_status_t LIN_MasterSendHeader (uint32_t *instance,* uint8_t *id*)

This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

instance	LPUART instance
id	Frame Identifier

Returns

An error code or lin_status_t

17.2.5.27 lin_status_t LIN_EnableIRQ (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.28 lin_status_t LIN_DisableIRQ (uint32_t *instance*)

Parameters

instance	LPUART instance

Returns

An error code or lin_status_t

17.2.5.29 void LIN_IRQHandler (uint8_t *instance*)

Parameters

instance	LPUART instance

Returns

void

17.2.5.30 lin_status_t LIN_AutoBaudCapture (uint32_t *instance*)

This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

instance	LPUART instance

Returns

lin_status_t

17.2.5.31 uint8_t LIN_ProcessParity (uint8_t *PID,* uint8_t *typeAction*)

If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID. This is not a public API as it is called by other API functions.

Parameters

PID	PID byte in case of checking parity bits or ID byte in case of making parity bits.
typeAction,:	1 for Checking parity bits, 0 for making parity bits

Returns

0xFF if parity bits are incorrect, ID in case of checking parity bits and they are correct. Function returns PID in case of making parity bits.

17.2.5.32 uint8_t LIN_MakeChecksumByte (const uint8_t ∗ *buffer,* uint8_t *sizeBuffer,* uint8_t *PID*)

Parameters

buffer	Pointer to Tx buffer
sizeBuffer	Number of bytes that are contained in the buffer.
PID	Protected Identifier byte.

Returns

the checksum byte.

17.2.6 Variable Documentation

17.2.6.1 const uint8_t* lin_state_t::txBuff

17.2.6.2 uint8_t* lin_state_t::rxBuff

17.2.6.3 uint8_t lin_state_t::cntByte

17.2.6.4 volatile uint8_t lin_state_t::txSize

17.2.6.5 volatile uint8_t lin_state_t::rxSize

17.2.6.6 uint8_t lin_state_t::checkSum

17.2.6.7 volatile bool lin_state_t::isTxBusy

17.2.6.8 volatile bool lin_state_t::isRxBusy

17.2.6.9 volatile bool lin_state_t::isTxBlocking

17.2.6.10 volatile bool lin_state_t::isRxBlocking

17.2.6.11 lin_callback_t lin_state_t::Callback

17.2.6.12 volatile bool lin_state_t::txCompleted

17.2.6.13 LPUART_Type* const g_linLpuartBase[FSL_FEATURE_SOC_LPUART_COUNT]

17.3 LIN LPUART Driver

17.3.1 Overview

Enumerations

- enum lin_lpuart_stop_bit_count_t {
  kLPUART_OneStopBit = 0U,
  kLPUART_TwoStopBit = 1U }
- enum _lin_lpuart_flags {
  kLPUART_TxDataRegEmptyFlag = (LPUART_STAT_TDRE_MASK),
  kLPUART_TransmissionCompleteFlag = (LPUART_STAT_TC_MASK),
  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 = (LPUART_STAT_FE_MASK),
  kLPUART_ParityErrorFlag = (LPUART_STAT_PF_MASK),
  kLPUART_LinBreakFlag = (LPUART_STAT_LBKDIF_MASK),
  kLPUART_RxActiveEdgeFlag = (LPUART_STAT_RXEDGIF_MASK),
  kLPUART_RxActiveFlag = (LPUART_STAT_RAF_MASK),
  kLPUART_DataMatch1Flag = LPUART_STAT_MA1F_MASK,
  kLPUART_DataMatch2Flag = LPUART_STAT_MA2F_MASK,
  kLPUART_NoiseErrorInRxDataRegFlag = (LPUART_DATA_NOISY_MASK >> 10),
  kLPUART_ParityErrorInRxDataRegFlag = (LPUART_DATA_PARITYE_MASK >> 10) }
- enum _lin_lpuart_interrupt_enable {
  kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIE_MASK >> 8),
  kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8),
  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) }
- enum _lin_lpuart_status {

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,
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) }

- enum lin_lpuart_bit_count_per_char_t {
  LPUART_8_BITS_PER_CHAR = 0x0U,
  LPUART_9_BITS_PER_CHAR = 0x1U,
  LPUART_10_BITS_PER_CHAR = 0x2U }

Functions

- static bool LIN_LPUART_GetRxDataPolarity (const LPUART_Type *base)
  Initializes an LIN_LPUART instance for LIN Network.
- lin_status_t LIN_LPUART_Deinit (LPUART_Type *base)
  Shuts down the LIN_LPUART by disabling interrupts and transmitter/receiver.
- lin_status_t LIN_LPUART_SendFrameDataBlocking (LPUART_Type *base, const uint8_t *tx-Buff, uint8_t txSize, uint32_t timeoutMSec)
  Sends Frame data out through the LIN_LPUART module using blocking method.
- lin_status_t LIN_LPUART_SendFrameData (LPUART_Type *base, const uint8_t *txBuff, uint8_t txSize)
  Sends frame data out through the LIN_LPUART module using non-blocking method.
- lin_status_t LIN_LPUART_GetTransmitStatus (LPUART_Type *base, uint8_t *bytesRemaining)
  Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission.
- lin_status_t LIN_LPUART_RecvFrmDataBlocking (LPUART_Type *base, uint8_t *rxBuff, uint8-_t rxSize, uint32_t timeoutMSec)
  Receives frame data through the LIN_LPUART module using blocking method.
- lin_status_t LIN_LPUART_RecvFrmData (LPUART_Type *base, uint8_t *rxBuff, uint8_t rxSize)
  Receives frame data through the LIN_LPUART module using non- blocking method.
- lin_status_t LIN_LPUART_AbortTransferData (LPUART_Type *base)
  Aborts an on-going non-blocking transmission/reception.
- lin_status_t LIN_LPUART_GetReceiveStatus (LPUART_Type *base, uint8_t *bytesRemaining)
  Get status of an on-going non-blocking reception While receiving frame data using non-blocking method, users can use this function to get status of that receiving.
- lin_status_t LIN_LPUART_GoToSleepMode (LPUART_Type *base)
  This function puts current node to sleep mode This function changes current node state to LIN_NODE_S-TATE_SLEEP_MODE.
- lin_status_t LIN_LPUART_GotoIdleState (LPUART_Type *base)

Puts current LIN node to Idle state This function changes current node state to LIN_NODE_STATE_IDLE.

- lin_status_t LIN_LPUART_SendWakeupSignal (LPUART_Type *base)
  Sends a wakeup signal through the LIN_LPUART interface.
- lin_status_t LIN_LPUART_MasterSendHeader (LPUART_Type *base, uint8_t id)
  Sends frame header out through the LIN_LPUART module using a non-blocking method.
- lin_status_t LIN_LPUART_EnableIRQ (LPUART_Type *base)
  Enables LIN_LPUART hardware interrupts.
- lin_status_t LIN_LPUART_DisableIRQ (LPUART_Type *base)
  Disables LIN_LPUART hardware interrupts.
- lin_status_t LIN_LPUART_AutoBaudCapture (uint32_t instance)
  This function capture bits time to detect break char, calculate baudrate from sync bits and enable transceiver if autobaud successful.
- void LIN_LPUART_IRQHandler (LPUART_Type *base)
  LIN_LPUART RX TX interrupt handler.

17.3.2 Enumeration Type Documentation

17.3.2.1 enum lin_lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

17.3.2.2 enum _lin_lpuart_flags

Enumerator

kLPUART_TxDataRegEmptyFlag Transmit data register empty flag, sets when transmit buffer is empty.
kLPUART_TransmissionCompleteFlag Transmission complete flag, sets when transmission activity complete.
kLPUART_RxDataRegFullFlag Receive data register full flag, sets when the receive data buffer is full.
kLPUART_IdleLineFlag Idle line detect flag, sets when idle line detected.
kLPUART_RxOverrunFlag Receive Overrun, sets when new data is received before data is read from receive register.
kLPUART_NoiseErrorFlag Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets
kLPUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected.
kLPUART_ParityErrorFlag If parity enabled, sets upon parity error detection.
kLPUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
kLPUART_RxActiveEdgeFlag Receive pin active edge interrupt flag, sets when active edge detected.

kLPUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start bit.
kLPUART_DataMatch1Flag The next character to be read from LPUART_DATA matches MA1.
kLPUART_DataMatch2Flag The next character to be read from LPUART_DATA matches MA2.
kLPUART_NoiseErrorInRxDataRegFlag NOISY bit, sets if noise detected in current data word.
kLPUART_ParityErrorInRxDataRegFlag PARITYE bit, sets if noise detected in current data word.

17.3.2.3 enum _lin_lpuart_interrupt_enable

Enumerator

kLPUART_LinBreakInterruptEnable LIN break detect.
kLPUART_RxActiveEdgeInterruptEnable Receive Active Edge.
kLPUART_TxDataRegEmptyInterruptEnable Transmit data register empty.
kLPUART_TransmissionCompleteInterruptEnable Transmission complete.
kLPUART_RxDataRegFullInterruptEnable Receiver data register full.
kLPUART_IdleLineInterruptEnable Idle line.
kLPUART_RxOverrunInterruptEnable Receiver Overrun.
kLPUART_NoiseErrorInterruptEnable Noise error flag.
kLPUART_FramingErrorInterruptEnable Framing error flag.
kLPUART_ParityErrorInterruptEnable Parity error flag.

17.3.2.4 enum _lin_lpuart_status

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_RxRingBufferOverrun LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverrun LPUART RX receiver overrun.
kStatus_LPUART_NoiseError LPUART noise error.
kStatus_LPUART_FramingError LPUART framing error.
kStatus_LPUART_ParityError LPUART parity error.

17.3.2.5 enum lin_lpuart_bit_count_per_char_t

Enumerator

LPUART_8_BITS_PER_CHAR 8-bit data characters
LPUART_9_BITS_PER_CHAR 9-bit data characters
LPUART_10_BITS_PER_CHAR 10-bit data characters

17.3.3 Function Documentation

17.3.3.1 static bool LIN_LPUART_GetRxDataPolarity (const LPUART_Type ∗ *base*) [inline], [static]

The caller provides memory for the driver state structures during initialization. The user must select the LIN_LPUART clock source in the application to initialize the LIN_LPUART. This function initializes a LPUART instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, set break field length to be 13 bit times minimum, enable the break detect interrupt, Rx complete interrupt, frame error detect interrupt, and enable the LPUART module transmitter and receiver

Parameters

instance	LIN_LPUART instance number
linUserConfig	user configuration structure of type lin_user_config_t
linCurrentState	pointer to the LIN_LPUART driver state structure

Returns

An error code or lin_status_t

17.3.3.2 lin_status_t LIN_LPUART_Deinit (LPUART_Type ∗ *base*)

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.3 lin_status_t LIN_LPUART_SendFrameDataBlocking (LPUART_Type * *base,* const uint8_t * *txBuff,* uint8_t *txSize,* uint32_t *timeoutMSec*)

This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete.

Parameters

instance	LIN_LPUART instance number
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send
timeoutMSec	timeout value in milli seconds

Returns

An error code or lin_status_t

17.3.3.4 lin_status_t LIN_LPUART_SendFrameData (LPUART_Type * *base,* const uint8_t * *txBuff,* uint8_t *txSize*)

This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data.

Parameters

instance	LIN_LPUART instance number
txBuff	source buffer containing 8-bit data chars to send
txSize	the number of bytes to send

Returns

An error code or lin_status_t

17.3.3.5 lin_status_t LIN_LPUART_GetTransmitStatus (LPUART_Type * *base,* uint8_t * *bytesRemaining*)

This function return LIN_TX_BUSY while sending, or LIN_TIMEOUT if timeout has occurred, or return LIN_SUCCESS when the transmission is complete. The bytesRemaining shows number of bytes that still needed to transmit.

Parameters

instance	LIN_LPUART instance number
bytes-Remaining	Number of bytes still needed to transmit

Returns

lin_status_t LIN_TX_BUSY, LIN_SUCCESS or LIN_TIMEOUT

17.3.3.6 lin_status_t LIN_LPUART_RecvFrmDataBlocking (LPUART_Type * *base,* uint8_t * *rxBuff,* uint8_t *rxSize,* uint32_t *timeoutMSec*)

This function will check the checksum byte. If the checksum is correct, it will receive the frame data. Blocking means that the function does not return until the reception is complete.

Parameters

instance	LIN_LPUART instance number
rxBuff	buffer containing 8-bit received data
rxSize	the number of bytes to receive
timeoutMSec	timeout value in milli seconds

Returns

An error code or lin_status_t

17.3.3.7 lin_status_t LIN_LPUART_RecvFrmData (LPUART_Type * *base,* uint8_t * *rxBuff,* uint8_t *rxSize*)

This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete.

Parameters

instance	LIN_LPUART instance number

rxBuff	buffer containing 8-bit received data
rxSize	the number of bytes to receive

Returns

An error code or lin_status_t

17.3.3.8 lin_status_t LIN_LPUART_AbortTransferData (LPUART_Type ∗ *base*)

While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.9 lin_status_t LIN_LPUART_GetReceiveStatus (LPUART_Type ∗ *base,* uint8_t ∗ *bytesRemaining*)

This function return the current event ID, LIN_RX_BUSY while receiving and return LIN_SUCCESS, or timeout (LIN_TIMEOUT) when the reception is complete. The bytesRemaining shows number of bytes that still needed to receive.

Parameters

instance	LIN_LPUART instance number
bytes-Remaining	Number of bytes still needed to receive

Returns

lin_status_t LIN_RX_BUSY, LIN_TIMEOUT or LIN_SUCCESS

17.3.3.10 lin_status_t LIN_LPUART_GoToSleepMode (LPUART_Type ∗ *base*)

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.11 lin_status_t LIN_LPUART_GotoIdleState (LPUART_Type * *base*)

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.12 lin_status_t LIN_LPUART_SendWakeupSignal (LPUART_Type * *base*)

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.13 lin_status_t LIN_LPUART_MasterSendHeader (LPUART_Type * *base,* uint8_t *id*)

This function sends LIN Break field, sync field then the ID with correct parity.

Parameters

instance	LIN_LPUART instance number
id	Frame Identifier

Returns

An error code or lin_status_t

17.3.3.14 lin_status_t LIN_LPUART_EnableIRQ (LPUART_Type * *base*)

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.15 lin_status_t LIN_LPUART_DisableIRQ (LPUART_Type * *base*)

Parameters

instance	LIN_LPUART instance number

Returns

An error code or lin_status_t

17.3.3.16 lin_status_t LIN_LPUART_AutoBaudCapture (uint32_t *instance*)

This function should only be used in Slave. The timer should be in mode input capture of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

Parameters

instance	LPUART instance

Returns

lin_status_t

17.3.3.17 void LIN_LPUART_IRQHandler (LPUART_Type * *base*)

Parameters

instance	LIN_LPUART instance number

Returns

void

Chapter 18
LLWU: Low-Leakage Wakeup Unit Driver

18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low-Leakage Wakeup Unit (LLWU) module of MCUXpresso SDK devices. The LLWU module allows the user to select external pin sources and internal modules as a wake-up source from low-leakage power modes.

18.2 External wakeup pins configurations

Configures the external wakeup pins' working modes, gets, and clears the wake pin flags. External wakeup pins are accessed by the `pinIndex`, which is started from 1. Numbers of the external pins depend on the SoC configuration.

18.3 Internal wakeup modules configurations

Enables/disables the internal wakeup modules and gets the module flags. Internal modules are accessed by `moduleIndex`, which is started from 1. Numbers of external pins depend the on SoC configuration.

18.4 Digital pin filter for external wakeup pin configurations

Configures the digital pin filter of the external wakeup pins' working modes, gets, and clears the pin filter flags. Digital pin filters are accessed by the `filterIndex`, which is started from 1. Numbers of external pins depend on the SoC configuration.

Data Structures

- struct llwu_external_pin_filter_mode_t
  An external input pin filter control structure. More...

Enumerations

- enum llwu_external_pin_mode_t {
  kLLWU_ExternalPinDisable = 0U,
  kLLWU_ExternalPinRisingEdge = 1U,
  kLLWU_ExternalPinFallingEdge = 2U,
  kLLWU_ExternalPinAnyEdge = 3U }
  External input pin control modes.
- enum llwu_pin_filter_mode_t {
  kLLWU_PinFilterDisable = 0U,
  kLLWU_PinFilterRisingEdge = 1U,
  kLLWU_PinFilterFallingEdge = 2U,
  kLLWU_PinFilterAnyEdge = 3U }
  Digital filter control modes.

Driver version

- #define FSL_LLWU_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  LLWU driver version 2.0.1.

Low-Leakage Wakeup Unit Control APIs

- void LLWU_SetExternalWakeupPinMode (LLWU_Type *base, uint32_t pinIndex, llwu_external_pin_mode_t pinMode)
  Sets the external input pin source mode.
- bool LLWU_GetExternalWakeupPinFlag (LLWU_Type *base, uint32_t pinIndex)
  Gets the external wakeup source flag.
- void LLWU_ClearExternalWakeupPinFlag (LLWU_Type *base, uint32_t pinIndex)
  Clears the external wakeup source flag.
- static void LLWU_EnableInternalModuleInterruptWakup (LLWU_Type *base, uint32_t moduleIndex, bool enable)
  Enables/disables the internal module source.
- static bool LLWU_GetInternalWakeupModuleFlag (LLWU_Type *base, uint32_t moduleIndex)
  Gets the external wakeup source flag.
- void LLWU_SetPinFilterMode (LLWU_Type *base, uint32_t filterIndex, llwu_external_pin_filter_mode_t filterMode)
  Sets the pin filter configuration.
- bool LLWU_GetPinFilterFlag (LLWU_Type *base, uint32_t filterIndex)
  Gets the pin filter configuration.
- void LLWU_ClearPinFilterFlag (LLWU_Type *base, uint32_t filterIndex)
  Clears the pin filter configuration.
- #define **INTERNAL_WAKEUP_MODULE_FLAG_REG** F3

18.5 Data Structure Documentation

18.5.1 struct llwu_external_pin_filter_mode_t

Data Fields

- uint32_t pinIndex
  A pin number.
- llwu_pin_filter_mode_t filterMode
  Filter mode.

18.6 Macro Definition Documentation

18.6.1 #define FSL_LLWU_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

18.7 Enumeration Type Documentation

18.7.1 enum llwu_external_pin_mode_t

Enumerator

kLLWU_ExternalPinDisable Pin disabled as a wakeup input.

kLLWU_ExternalPinRisingEdge Pin enabled with the rising edge detection.
kLLWU_ExternalPinFallingEdge Pin enabled with the falling edge detection.
kLLWU_ExternalPinAnyEdge Pin enabled with any change detection.

18.7.2 enum llwu_pin_filter_mode_t

Enumerator

kLLWU_PinFilterDisable Filter disabled.
kLLWU_PinFilterRisingEdge Filter positive edge detection.
kLLWU_PinFilterFallingEdge Filter negative edge detection.
kLLWU_PinFilterAnyEdge Filter any edge detection.

18.8 Function Documentation

18.8.1 void LLWU_SetExternalWakeupPinMode (LLWU_Type * *base*, uint32_t *pinIndex*, llwu_external_pin_mode_t *pinMode*)

This function sets the external input pin source mode that is used as a wake up source.

Parameters

base	LLWU peripheral base address.
pinIndex	A pin index to be enabled as an external wakeup source starting from 1.
pinMode	A pin configuration mode defined in the llwu_external_pin_modes_t.

18.8.2 bool LLWU_GetExternalWakeupPinFlag (LLWU_Type * *base*, uint32_t *pinIndex*)

This function checks the external pin flag to detect whether the MCU is woken up by the specific pin.

Parameters

base	LLWU peripheral base address.
pinIndex	A pin index, which starts from 1.

Returns

True if the specific pin is a wakeup source.

18.8.3 void LLWU_ClearExternalWakeupPinFlag (LLWU_Type * *base,* uint32_t *pinIndex*)

This function clears the external wakeup source flag for a specific pin.

Parameters

base	LLWU peripheral base address.
pinIndex	A pin index, which starts from 1.

18.8.4 static void LLWU_EnableInternalModuleInterruptWakup (LLWU_Type * *base,* uint32_t *moduleIndex,* bool *enable*) [inline], [static]

This function enables/disables the internal module source mode that is used as a wake up source.

Parameters

base	LLWU peripheral base address.
moduleIndex	A module index to be enabled as an internal wakeup source starting from 1.
enable	An enable or a disable setting

18.8.5 static bool LLWU_GetInternalWakeupModuleFlag (LLWU_Type * *base,* uint32_t *moduleIndex*) [inline], [static]

This function checks the external pin flag to detect whether the system is woken up by the specific pin.

Parameters

base	LLWU peripheral base address.
moduleIndex	A module index, which starts from 1.

Returns

True if the specific pin is a wake up source.

18.8.6 void LLWU_SetPinFilterMode (LLWU_Type * *base,* uint32_t *filterIndex,* llwu_external_pin_filter_mode_t *filterMode*)

This function sets the pin filter configuration.

Parameters

base	LLWU peripheral base address.
filterIndex	A pin filter index used to enable/disable the digital filter, starting from 1.
filterMode	A filter mode configuration

18.8.7 bool LLWU_GetPinFilterFlag (LLWU_Type * *base,* uint32_t *filterIndex*)

This function gets the pin filter flag.

Parameters

base	LLWU peripheral base address.
filterIndex	A pin filter index, which starts from 1.

Returns

True if the flag is a source of the existing low-leakage power mode.

18.8.8 void LLWU_ClearPinFilterFlag (LLWU_Type * *base,* uint32_t *filterIndex*)

This function clears the pin filter flag.

Parameters

base	LLWU peripheral base address.
filterIndex	A pin filter index to clear the flag, starting from 1.

Chapter 19
LPTMR: Low-Power Timer

19.1 Overview

The MCUXpresso SDK provides a driver for the Low-Power Timer (LPTMR) of MCUXpresso SDK devices.

19.2 Function groups

The LPTMR driver supports operating the module as a time counter or as a pulse counter.

19.2.1 Initialization and deinitialization

The function LPTMR_Init() initializes the LPTMR with specified configurations. The function LPTMR_-GetDefaultConfig() gets the default configurations. The initialization function configures the LPTMR for a timer or a pulse counter mode mode. It also sets up the LPTMR's free running mode operation and a clock source.

The function LPTMR_DeInit() disables the LPTMR module and gates the module clock.

19.2.2 Timer period Operations

The function LPTMR_SetTimerPeriod() sets the timer period in units of count. Timers counts from 0 to the count value set here.

The function LPTMR_GetCurrentTimerCount() reads the current timer counting value. This function returns the real-time timer counting value ranging from 0 to a timer period.

The timer period operation function takes the count value in ticks. Call the utility macros provided in the fsl_common.h file to convert to microseconds or milliseconds.

19.2.3 Start and Stop timer operations

The function LPTMR_StartTimer() starts the timer counting. After calling this function, the timer counts up to the counter value set earlier by using the LPTMR_SetPeriod() function. Each time the timer reaches the count value and increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

The function LPTMR_StopTimer() stops the timer counting and resets the timer's counter register.

19.2.4 Status

Provides functions to get and clear the LPTMR status.

19.2.5 Interrupt

Provides functions to enable/disable LPTMR interrupts and get the currently enabled interrupts.

19.3 Typical use case

19.3.1 LPTMR tick example

Updates the LPTMR period and toggles an LED periodically.

```
int main(void)
{
    uint32_t currentCounter = 0U;
    lptmr_config_t lptmrConfig;

    LED_INIT();

    /* Board pin, clock, debug console initialization */
    BOARD_InitHardware();

    /* Configures the LPTMR */
    LPTMR_GetDefaultConfig(&lptmrConfig);

    /* Initializes the LPTMR */
    LPTMR_Init(LPTMR0, &lptmrConfig);

    /* Sets the timer period */
    LPTMR_SetTimerPeriod(LPTMR0, USEC_TO_COUNT(1000000U, LPTMR_SOURCE_CLOCK));

    /* Enables a timer interrupt */
    LPTMR_EnableInterrupts(LPTMR0,
      kLPTMR_TimerInterruptEnable);

    /* Enables the NVIC */
    EnableIRQ(LPTMR0_IRQn);

    PRINTF("Low Power Timer Example\r\n");

    /* Starts counting */
    LPTMR_StartTimer(LPTMR0);
    while (1)
    {
        if (currentCounter != lptmrCounter)
        {
            currentCounter = lptmrCounter;
            PRINTF("LPTMR interrupt No.%d \r\n", currentCounter);
        }
    }
}
```

Data Structures

- struct lptmr_config_t
  LPTMR config structure. More...

Enumerations

- enum lptmr_pin_select_t {
  kLPTMR_PinSelectInput_0 = 0x0U,
  kLPTMR_PinSelectInput_1 = 0x1U,
  kLPTMR_PinSelectInput_2 = 0x2U,
  kLPTMR_PinSelectInput_3 = 0x3U }
  LPTMR pin selection used in pulse counter mode.
- enum lptmr_pin_polarity_t {
  kLPTMR_PinPolarityActiveHigh = 0x0U,
  kLPTMR_PinPolarityActiveLow = 0x1U }
  LPTMR pin polarity used in pulse counter mode.
- enum lptmr_timer_mode_t {
  kLPTMR_TimerModeTimeCounter = 0x0U,
  kLPTMR_TimerModePulseCounter = 0x1U }
  LPTMR timer mode selection.
- enum lptmr_prescaler_glitch_value_t {
  kLPTMR_Prescale_Glitch_0 = 0x0U,
  kLPTMR_Prescale_Glitch_1 = 0x1U,
  kLPTMR_Prescale_Glitch_2 = 0x2U,
  kLPTMR_Prescale_Glitch_3 = 0x3U,
  kLPTMR_Prescale_Glitch_4 = 0x4U,
  kLPTMR_Prescale_Glitch_5 = 0x5U,
  kLPTMR_Prescale_Glitch_6 = 0x6U,
  kLPTMR_Prescale_Glitch_7 = 0x7U,
  kLPTMR_Prescale_Glitch_8 = 0x8U,
  kLPTMR_Prescale_Glitch_9 = 0x9U,
  kLPTMR_Prescale_Glitch_10 = 0xAU,
  kLPTMR_Prescale_Glitch_11 = 0xBU,
  kLPTMR_Prescale_Glitch_12 = 0xCU,
  kLPTMR_Prescale_Glitch_13 = 0xDU,
  kLPTMR_Prescale_Glitch_14 = 0xEU,
  kLPTMR_Prescale_Glitch_15 = 0xFU }
  LPTMR prescaler/glitch filter values.
- enum lptmr_prescaler_clock_select_t {
  kLPTMR_PrescalerClock_0 = 0x0U,
  kLPTMR_PrescalerClock_1 = 0x1U,
  kLPTMR_PrescalerClock_2 = 0x2U,
  kLPTMR_PrescalerClock_3 = 0x3U }
  LPTMR prescaler/glitch filter clock select.
- enum lptmr_interrupt_enable_t { kLPTMR_TimerInterruptEnable = LPTMR_CSR_TIE_MASK }
  List of the LPTMR interrupts.
- enum lptmr_status_flags_t { kLPTMR_TimerCompareFlag = LPTMR_CSR_TCF_MASK }
  List of the LPTMR status flags.

Driver version

- #define FSL_LPTMR_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Version 2.0.1.

Initialization and deinitialization

- void LPTMR_Init (LPTMR_Type ∗base, const lptmr_config_t ∗config)
  Ungates the LPTMR clock and configures the peripheral for a basic operation.
- void LPTMR_Deinit (LPTMR_Type ∗base)
  Gates the LPTMR clock.
- void LPTMR_GetDefaultConfig (lptmr_config_t ∗config)
  Fills in the LPTMR configuration structure with default settings.

Interrupt Interface

- static void LPTMR_EnableInterrupts (LPTMR_Type ∗base, uint32_t mask)
  Enables the selected LPTMR interrupts.
- static void LPTMR_DisableInterrupts (LPTMR_Type ∗base, uint32_t mask)
  Disables the selected LPTMR interrupts.
- static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type ∗base)
  Gets the enabled LPTMR interrupts.

Status Interface

- static uint32_t LPTMR_GetStatusFlags (LPTMR_Type ∗base)
  Gets the LPTMR status flags.
- static void LPTMR_ClearStatusFlags (LPTMR_Type ∗base, uint32_t mask)
  Clears the LPTMR status flags.

Read and write the timer period

- static void LPTMR_SetTimerPeriod (LPTMR_Type ∗base, uint32_t ticks)
  Sets the timer period in units of count.
- static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type ∗base)
  Reads the current timer counting value.

Timer Start and Stop

- static void LPTMR_StartTimer (LPTMR_Type ∗base)
  Starts the timer.
- static void LPTMR_StopTimer (LPTMR_Type ∗base)
  Stops the timer.

19.4 Data Structure Documentation

19.4.1 struct lptmr_config_t

This structure holds the configuration settings for the LPTMR peripheral. To initialize this structure to reasonable defaults, call the LPTMR_GetDefaultConfig() function and pass a pointer to your configuration structure instance.

The configuration struct can be made constant so it resides in flash.

Data Fields

- lptmr_timer_mode_t timerMode
  Time counter mode or pulse counter mode.
- lptmr_pin_select_t pinSelect
  LPTMR pulse input pin select; used only in pulse counter mode.
- lptmr_pin_polarity_t pinPolarity
  LPTMR pulse input pin polarity; used only in pulse counter mode.
- bool enableFreeRunning
  True: enable free running, counter is reset on overflow False: counter is reset when the compare flag is set.
- bool bypassPrescaler
  True: bypass prescaler; false: use clock from prescaler.
- lptmr_prescaler_clock_select_t prescalerClockSource
  LPTMR clock source.
- lptmr_prescaler_glitch_value_t value
  Prescaler or glitch filter value.

19.5 Enumeration Type Documentation

19.5.1 enum lptmr_pin_select_t

Enumerator

kLPTMR_PinSelectInput_0 Pulse counter input 0 is selected.
kLPTMR_PinSelectInput_1 Pulse counter input 1 is selected.
kLPTMR_PinSelectInput_2 Pulse counter input 2 is selected.
kLPTMR_PinSelectInput_3 Pulse counter input 3 is selected.

19.5.2 enum lptmr_pin_polarity_t

Enumerator

kLPTMR_PinPolarityActiveHigh Pulse Counter input source is active-high.
kLPTMR_PinPolarityActiveLow Pulse Counter input source is active-low.

19.5.3 enum lptmr_timer_mode_t

Enumerator

kLPTMR_TimerModeTimeCounter Time Counter mode.
kLPTMR_TimerModePulseCounter Pulse Counter mode.

19.5.4 enum lptmr_prescaler_glitch_value_t

Enumerator

kLPTMR_Prescale_Glitch_0 Prescaler divide 2, glitch filter does not support this setting.
kLPTMR_Prescale_Glitch_1 Prescaler divide 4, glitch filter 2.
kLPTMR_Prescale_Glitch_2 Prescaler divide 8, glitch filter 4.
kLPTMR_Prescale_Glitch_3 Prescaler divide 16, glitch filter 8.
kLPTMR_Prescale_Glitch_4 Prescaler divide 32, glitch filter 16.
kLPTMR_Prescale_Glitch_5 Prescaler divide 64, glitch filter 32.
kLPTMR_Prescale_Glitch_6 Prescaler divide 128, glitch filter 64.
kLPTMR_Prescale_Glitch_7 Prescaler divide 256, glitch filter 128.
kLPTMR_Prescale_Glitch_8 Prescaler divide 512, glitch filter 256.
kLPTMR_Prescale_Glitch_9 Prescaler divide 1024, glitch filter 512.
kLPTMR_Prescale_Glitch_10 Prescaler divide 2048 glitch filter 1024.
kLPTMR_Prescale_Glitch_11 Prescaler divide 4096, glitch filter 2048.
kLPTMR_Prescale_Glitch_12 Prescaler divide 8192, glitch filter 4096.
kLPTMR_Prescale_Glitch_13 Prescaler divide 16384, glitch filter 8192.
kLPTMR_Prescale_Glitch_14 Prescaler divide 32768, glitch filter 16384.
kLPTMR_Prescale_Glitch_15 Prescaler divide 65536, glitch filter 32768.

19.5.5 enum lptmr_prescaler_clock_select_t

Note

Clock connections are SoC-specific

Enumerator

kLPTMR_PrescalerClock_0 Prescaler/glitch filter clock 0 selected.
kLPTMR_PrescalerClock_1 Prescaler/glitch filter clock 1 selected.
kLPTMR_PrescalerClock_2 Prescaler/glitch filter clock 2 selected.
kLPTMR_PrescalerClock_3 Prescaler/glitch filter clock 3 selected.

19.5.6 enum lptmr_interrupt_enable_t

Enumerator

kLPTMR_TimerInterruptEnable Timer interrupt enable.

19.5.7 enum lptmr_status_flags_t

Enumerator

kLPTMR_TimerCompareFlag Timer compare flag.

19.6 Function Documentation

19.6.1 void LPTMR_Init (LPTMR_Type ∗ *base,* const lptmr_config_t ∗ *config*)

Note

This API should be called at the beginning of the application using the LPTMR driver.

Parameters

base	LPTMR peripheral base address
config	A pointer to the LPTMR configuration structure.

19.6.2 void LPTMR_Deinit (LPTMR_Type ∗ *base*)

Parameters

base	LPTMR peripheral base address

19.6.3 void LPTMR_GetDefaultConfig (lptmr_config_t ∗ *config*)

The default values are as follows.

```
*    config->timerMode = kLPTMR_TimerModeTimeCounter;
*    config->pinSelect = kLPTMR_PinSelectInput_0;
*    config->pinPolarity = kLPTMR_PinPolarityActiveHigh;
*    config->enableFreeRunning = false;
*    config->bypassPrescaler = true;
*    config->prescalerClockSource = kLPTMR_PrescalerClock_1;
*    config->value = kLPTMR_Prescale_Glitch_0;
*
```

Parameters

config	A pointer to the LPTMR configuration structure.

19.6.4 static void LPTMR_EnableInterrupts (LPTMR_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	LPTMR peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration lptmr_interrupt_enable_t

19.6.5 static void LPTMR_DisableInterrupts (LPTMR_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	LPTMR peripheral base address
mask	The interrupts to disable. This is a logical OR of members of the enumeration lptmr_interrupt_enable_t.

19.6.6 static uint32_t LPTMR_GetEnabledInterrupts (LPTMR_Type * *base*) [inline], [static]

Parameters

base	LPTMR peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration lptmr_interrupt_enable_t

19.6.7 static uint32_t LPTMR_GetStatusFlags (LPTMR_Type * *base*) [inline], [static]

Parameters

base	LPTMR peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration lptmr_status_flags_t

19.6.8 static void LPTMR_ClearStatusFlags (LPTMR_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	LPTMR peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration lptmr_-status_flags_t.

19.6.9 static void LPTMR_SetTimerPeriod (LPTMR_Type * *base*, uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the CMR register.

Note

1. The TCF flag is set with the CNR equals the count provided here and then increments.
2. Call the utility macros provided in the fsl_common.h to convert to ticks.

Parameters

base	LPTMR peripheral base address
ticks	A timer period in units of ticks, which should be equal or greater than 1.

19.6.10 static uint32_t LPTMR_GetCurrentTimerCount (LPTMR_Type * *base*) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

base	LPTMR peripheral base address

Returns

The current counter value in ticks

19.6.11 static void LPTMR_StartTimer (LPTMR_Type * *base*) [inline], [static]

After calling this function, the timer counts up to the CMR register value. Each time the timer reaches the CMR value and then increments, it generates a trigger pulse and sets the timeout interrupt flag. An interrupt is also triggered if the timer interrupt is enabled.

Parameters

base	LPTMR peripheral base address

19.6.12 static void LPTMR_StopTimer (LPTMR_Type * *base*) [inline], [static]

This function stops the timer and resets the timer's counter register.

Parameters

base	LPTMR peripheral base address

Chapter 20
LPUART: Low Power UART Driver

20.1 Overview

Modules

- LPUART DMA Driver
- LPUART Driver
- LPUART FreeRTOS Driver
- LPUART eDMA Driver

20.2 LPUART Driver

20.2.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Low Power UART (LPUART) module of MCUXpresso SDK devices.

20.2.2 Typical use case

20.2.2.1 LPUART Operation

```
uint8_t ch;
LPUART_GetDefaultConfig(&user_config);
user_config.baudRate = 115200U;
config.enableTx = true;
config.enableRx = true;

LPUART_Init(LPUART1,&user_config,120000000U);

LPUART_WriteBlocking(LPUART1, txbuff, sizeof(txbuff) - 1);

while(1)
{
    LPUART_ReadBlocking(LPUART1, &ch, 1);
    LPUART_WriteBlocking(LPUART1, &ch, 1);
}
```

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...

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 _lpuart_status {
  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 }
  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 }
  LPUART data bits count.
- enum lpuart_stop_bit_count_t {
  kLPUART_OneStopBit = 0U,
  kLPUART_TwoStopBit = 1U }
  LPUART stop bit count.
- enum _lpuart_interrupt_enable {
  kLPUART_LinBreakInterruptEnable = (LPUART_BAUD_LBKDIE_MASK >> 8),
  kLPUART_RxActiveEdgeInterruptEnable = (LPUART_BAUD_RXEDGIE_MASK >> 8),
  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) }
  LPUART interrupt configuration structure, default settings all disabled.
- enum _lpuart_flags {

kLPUART_TxDataRegEmptyFlag,
kLPUART_TransmissionCompleteFlag,
kLPUART_RxDataRegFullFlag,
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,
kLPUART_RxActiveFlag,
kLPUART_DataMatch1Flag = LPUART_STAT_MA1F_MASK,
kLPUART_DataMatch2Flag = LPUART_STAT_MA2F_MASK,
kLPUART_NoiseErrorInRxDataRegFlag,
kLPUART_ParityErrorInRxDataRegFlag }

LPUART status flags.

Driver version

- #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))
  LPUART driver version 2.2.3.

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.
- status_t LPUART_SetBaudRate (LPUART_Type *base, uint32_t baudRate_Bps, uint32_t srcClock_Hz)
  Sets the LPUART instance baudrate.

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.
- 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

- 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.
- void 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.
- 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 written to the LPUART transmitter register.

- 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, lpuart_handle_t *handle)
  LPUART IRQ handle function.
- void LPUART_TransferHandleErrorIRQ (LPUART_Type *base, lpuart_handle_t *handle)
  LPUART Error IRQ handle function.

20.2.3 Data Structure Documentation

20.2.3.1 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.
- bool enableTx
  Enable TX.
- bool enableRx
  Enable RX.

20.2.3.2 struct lpuart_transfer_t

Data Fields

- uint8_t * data
  The buffer of data to be transfer.
- size_t dataSize
  The byte count to be transfer.

20.2.3.2.0.58 Field Documentation

20.2.3.2.0.58.1 uint8_t* lpuart_transfer_t::data

20.2.3.2.0.58.2 size_t lpuart_transfer_t::dataSize

20.2.3.3 struct _lpuart_handle

Data Fields

- uint8_t *volatile txData
  Address of remaining data to send.
- volatile size_t txDataSize
  Size of the remaining data to send.
- size_t txDataSizeAll
  Size of the data to send out.
- uint8_t *volatile rxData
  Address of remaining data to receive.
- volatile size_t rxDataSize
  Size of the remaining data to receive.
- size_t rxDataSizeAll
  Size of the data to receive.
- uint8_t * rxRingBuffer
  Start address of the receiver ring buffer.
- size_t rxRingBufferSize
  Size of the ring buffer.
- volatile uint16_t rxRingBufferHead
  Index for the driver to store received data into ring buffer.
- volatile uint16_t rxRingBufferTail
  Index for the user to get data from the ring buffer.
- 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.

20.2.3.3.0.59 Field Documentation

20.2.3.3.0.59.1 uint8_t∗ volatile lpuart_handle_t::txData

20.2.3.3.0.59.2 volatile size_t lpuart_handle_t::txDataSize

20.2.3.3.0.59.3 size_t lpuart_handle_t::txDataSizeAll

20.2.3.3.0.59.4 uint8_t∗ volatile lpuart_handle_t::rxData

20.2.3.3.0.59.5 volatile size_t lpuart_handle_t::rxDataSize

20.2.3.3.0.59.6 size_t lpuart_handle_t::rxDataSizeAll

20.2.3.3.0.59.7 uint8_t∗ lpuart_handle_t::rxRingBuffer

20.2.3.3.0.59.8 size_t lpuart_handle_t::rxRingBufferSize

20.2.3.3.0.59.9 volatile uint16_t lpuart_handle_t::rxRingBufferHead

20.2.3.3.0.59.10 volatile uint16_t lpuart_handle_t::rxRingBufferTail

20.2.3.3.0.59.11 lpuart_transfer_callback_t lpuart_handle_t::callback

20.2.3.3.0.59.12 void∗ lpuart_handle_t::userData

20.2.3.3.0.59.13 volatile uint8_t lpuart_handle_t::txState

20.2.3.3.0.59.14 volatile uint8_t lpuart_handle_t::rxState

20.2.4 Macro Definition Documentation

20.2.4.1 #define FSL_LPUART_DRIVER_VERSION (MAKE_VERSION(2, 2, 3))

20.2.5 Typedef Documentation

20.2.5.1 typedef void(∗ lpuart_transfer_callback_t)(LPUART_Type ∗base, lpuart_handle_t ∗handle, status_t status, void ∗userData)

20.2.6 Enumeration Type Documentation

20.2.6.1 enum _lpuart_status

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_RxRingBufferOverrun LPUART RX software ring buffer overrun.
kStatus_LPUART_RxHardwareOverrun LPUART RX receiver overrun.
kStatus_LPUART_NoiseError LPUART noise error.
kStatus_LPUART_FramingError LPUART framing error.
kStatus_LPUART_ParityError LPUART parity error.
kStatus_LPUART_BaudrateNotSupport Baudrate is not support in current clock source.

20.2.6.2 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.

20.2.6.3 enum lpuart_data_bits_t

Enumerator

kLPUART_EightDataBits Eight data bit.

20.2.6.4 enum lpuart_stop_bit_count_t

Enumerator

kLPUART_OneStopBit One stop bit.
kLPUART_TwoStopBit Two stop bits.

20.2.6.5 enum _lpuart_interrupt_enable

This structure contains the settings for all LPUART interrupt configurations.

Enumerator

kLPUART_LinBreakInterruptEnable LIN break detect.
kLPUART_RxActiveEdgeInterruptEnable Receive Active Edge.
kLPUART_TxDataRegEmptyInterruptEnable Transmit data register empty.

kLPUART_TransmissionCompleteInterruptEnable Transmission complete.
kLPUART_RxDataRegFullInterruptEnable Receiver data register full.
kLPUART_IdleLineInterruptEnable Idle line.
kLPUART_RxOverrunInterruptEnable Receiver Overrun.
kLPUART_NoiseErrorInterruptEnable Noise error flag.
kLPUART_FramingErrorInterruptEnable Framing error flag.
kLPUART_ParityErrorInterruptEnable Parity error flag.

20.2.6.6 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.
kLPUART_TransmissionCompleteFlag Transmission complete flag, sets when transmission activity complete.
kLPUART_RxDataRegFullFlag Receive data register full flag, sets when the receive data buffer is full.
kLPUART_IdleLineFlag Idle line detect flag, sets when idle line detected.
kLPUART_RxOverrunFlag Receive Overrun, sets when new data is received before data is read from receive register.
kLPUART_NoiseErrorFlag Receive takes 3 samples of each received bit. If any of these samples differ, noise flag sets
kLPUART_FramingErrorFlag Frame error flag, sets if logic 0 was detected where stop bit expected.
kLPUART_ParityErrorFlag If parity enabled, sets upon parity error detection.
kLPUART_LinBreakFlag LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.
kLPUART_RxActiveEdgeFlag Receive pin active edge interrupt flag, sets when active edge detected.
kLPUART_RxActiveFlag Receiver Active Flag (RAF), sets at beginning of valid start bit.
kLPUART_DataMatch1Flag The next character to be read from LPUART_DATA matches MA1.
kLPUART_DataMatch2Flag The next character to be read from LPUART_DATA matches MA2.
kLPUART_NoiseErrorInRxDataRegFlag NOISY bit, sets if noise detected in current data word.
kLPUART_ParityErrorInRxDataRegFlag PARITYE bit, sets if noise detected in current data word.

20.2.7 Function Documentation

20.2.7.1 status_t LPUART_Init (LPUART_Type * *base*, const lpuart_config_t * *config*, uint32_t *srcClock_Hz*)

This function configures the LPUART module with user-defined settings. Call the LPUART_GetDefaultConfig() function to configure the configuration structure and get the default configuration. The example below shows how to use this API to configure the LPUART.

```
*  lpuart_config_t lpuartConfig;
*  lpuartConfig.baudRate_Bps = 115200U;
*  lpuartConfig.parityMode = kLPUART_ParityDisabled;
*  lpuartConfig.dataBitsCount = kLPUART_EightDataBits;
*  lpuartConfig.isMsb = false;
*  lpuartConfig.stopBitCount = kLPUART_OneStopBit;
*  lpuartConfig.txFifoWatermark = 0;
*  lpuartConfig.rxFifoWatermark = 1;
*  LPUART_Init(LPUART1, &lpuartConfig, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
config	Pointer to a user-defined configuration structure.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_BaudrateNotSupport	Baudrate is not support in current clock source.
kStatus_Success	LPUART initialize succeed

20.2.7.2 void LPUART_Deinit (LPUART_Type * *base*)

This function waits for transmit to complete, disables TX and RX, and disables the LPUART clock.

Parameters

base	LPUART peripheral base address.

20.2.7.3 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->enableTx = false; lpuartConfig->enableRx = false;

Parameters

config	Pointer to a configuration structure.

20.2.7.4 status_t LPUART_SetBaudRate (LPUART_Type * *base*, uint32_t *baudRate_Bps*, uint32_t *srcClock_Hz*)

This function configures the LPUART module baudrate. This function is used to update the LPUART module baudrate after the LPUART module is initialized by the LPUART_Init.

```
*  LPUART_SetBaudRate(LPUART1, 115200U, 20000000U);
*
```

Parameters

base	LPUART peripheral base address.
baudRate_Bps	LPUART baudrate to be set.
srcClock_Hz	LPUART clock source frequency in HZ.

Return values

kStatus_LPUART_-BaudrateNotSupport	Baudrate is not supported in the current clock source.
kStatus_Success	Set baudrate succeeded.

20.2.7.5 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

base	LPUART peripheral base address.

Returns

LPUART status flags which are ORed by the enumerators in the _lpuart_flags.

20.2.7.6 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 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

base	LPUART peripheral base address.
mask	the status flags to be cleared. The user can use the enumerators in the _lpuart_status_flag_t to do the OR operation and get the mask.

Returns

0 succeed, others failed.

Return values

kStatus_LPUART_FlagCannotClearManually	The flag can't be cleared by this function but it is cleared automatically by hardware.
kStatus_Success	Status in the mask are cleared.

20.2.7.7 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

base	LPUART peripheral base address.
mask	The interrupts to enable. Logical OR of _uart_interrupt_enable.

20.2.7.8 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

base	LPUART peripheral base address.
mask	The interrupts to disable. Logical OR of _lpuart_interrupt_enable.

20.2.7.9 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

base	LPUART peripheral base address.

Returns

LPUART interrupt flags which are logical OR of the enumerators in _lpuart_interrupt_enable.

20.2.7.10 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

base	LPUART peripheral base address.

Returns

LPUART data register addresses which are used both by the transmitter and receiver.

20.2.7.11 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

base	LPUART peripheral base address.
enable	True to enable, false to disable.

20.2.7.12 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

base	LPUART peripheral base address.
enable	True to enable, false to disable.

20.2.7.13 static void LPUART_EnableTx (LPUART_Type * *base,* bool *enable*) [inline], [static]

This function enables or disables the LPUART transmitter.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

20.2.7.14 static void LPUART_EnableRx (LPUART_Type * *base,* bool *enable*) [inline], [static]

This function enables or disables the LPUART receiver.

Parameters

base	LPUART peripheral base address.
enable	True to enable, false to disable.

20.2.7.15 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

base	LPUART peripheral base address.
data	Data write to the TX register.

20.2.7.16 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

base	LPUART peripheral base address.

Returns

Data read from data register.

20.2.7.17 void LPUART_WriteBlocking (LPUART_Type * *base,* const uint8_t * *data,* size_t *length*)

This function polls the transmitter register, waits for the register to be empty or for TX FIFO to have room, and writes data to the transmitter buffer.

Note

This function does not check whether all data has been sent out to the bus. Before disabling the transmitter, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

base	LPUART peripheral base address.
data	Start address of the data to write.
length	Size of the data to write.

20.2.7.18 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

base	LPUART peripheral base address.
data	Start address of the buffer to store the received data.
length	Size of the buffer.

Return values

kStatus_LPUART_RxHardwareOverrun	Receiver overrun happened while receiving data.
kStatus_LPUART_NoiseError	Noise error happened while receiving data.
kStatus_LPUART_FramingError	Framing error happened while receiving data.
kStatus_LPUART_ParityError	Parity error happened while receiving data.
kStatus_Success	Successfully received all data.

20.2.7.19 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.
callback	Callback function.
userData	User data.

20.2.7.20 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 TX, check the kLPUART_TransmissionCompleteFlag to ensure that the transmit is finished.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	Successfully start the data transmission.
kStatus_LPUART_TxBusy	Previous transmission still not finished, data not all written to the TX register.
kStatus_InvalidArgument	Invalid argument.

20.2.7.21 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.
ringBuffer	Start address of ring buffer for background receiving. Pass NULL to disable the ring buffer.
ringBufferSize	size of the ring buffer.

20.2.7.22 void LPUART_TransferStopRingBuffer (LPUART_Type * *base,* lpuart_handle_t * *handle*)

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

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

20.2.7.23 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.

20.2.7.24 status_t LPUART_TransferGetSendCount (LPUART_Type * *base*, lpuart_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been written to LPUART TX register by an interrupt method.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferInProgress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

20.2.7.25 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART transfer structure, see #uart_transfer_t.
receivedBytes	Bytes received from the ring buffer directly.

Return values

kStatus_Success	Successfully queue the transfer into the transmit queue.
kStatus_LPUART_RxBusy	Previous receive request is not finished.
kStatus_InvalidArgument	Invalid argument.

20.2.7.26 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.

20.2.7.27 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferInProgress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

20.2.7.28 void LPUART_TransferHandleIRQ (LPUART_Type * *base*, lpuart_handle_t * *handle*)

This function handles the LPUART transmit and receive IRQ request.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

20.2.7.29 void LPUART_TransferHandleErrorIRQ (LPUART_Type * *base*, lpuart_handle_t * *handle*)

This function handles the LPUART error IRQ request.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.

20.3 LPUART DMA Driver

20.3.1 Overview

Data Structures

- struct lpuart_dma_handle_t
  LPUART DMA handle. More...

Typedefs

- typedef void(∗ lpuart_dma_transfer_callback_t)(LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, status_t status, void ∗userData)
  LPUART transfer callback function.

EDMA transactional

- void LPUART_TransferCreateHandleDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_dma_transfer_callback_t callback, void ∗userData, dma_handle_t ∗txDmaHandle, dma_handle_t ∗rxDmaHandle)
  Initializes the LPUART handle which is used in transactional functions.
- status_t LPUART_TransferSendDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Sends data using DMA.
- status_t LPUART_TransferReceiveDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, lpuart_transfer_t ∗xfer)
  Receives data using DMA.
- void LPUART_TransferAbortSendDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle)
  Aborts the sent data using DMA.
- void LPUART_TransferAbortReceiveDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle)
  Aborts the received data using DMA.
- status_t LPUART_TransferGetSendCountDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of bytes written to the LPUART TX register.
- status_t LPUART_TransferGetReceiveCountDMA (LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, uint32_t ∗count)
  Gets the number of received bytes.

20.3.2 Data Structure Documentation

20.3.2.1 struct _lpuart_dma_handle

Data Fields

- lpuart_dma_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.
- dma_handle_t ∗ txDmaHandle

  The DMA TX channel used.
- dma_handle_t ∗ rxDmaHandle

  The DMA RX channel used.
- volatile uint8_t txState

  TX transfer state.
- volatile uint8_t rxState

  RX transfer state.

20.3.2.1.0.60 Field Documentation

20.3.2.1.0.60.1 lpuart_dma_transfer_callback_t lpuart_dma_handle_t::callback

20.3.2.1.0.60.2 void∗ lpuart_dma_handle_t::userData

20.3.2.1.0.60.3 size_t lpuart_dma_handle_t::rxDataSizeAll

20.3.2.1.0.60.4 size_t lpuart_dma_handle_t::txDataSizeAll

20.3.2.1.0.60.5 dma_handle_t∗ lpuart_dma_handle_t::txDmaHandle

20.3.2.1.0.60.6 dma_handle_t∗ lpuart_dma_handle_t::rxDmaHandle

20.3.2.1.0.60.7 volatile uint8_t lpuart_dma_handle_t::txState

20.3.3 Typedef Documentation

20.3.3.1 typedef void(∗ lpuart_dma_transfer_callback_t)(LPUART_Type ∗base, lpuart_dma_handle_t ∗handle, status_t status, void ∗userData)

20.3.4 Function Documentation

20.3.4.1 void LPUART_TransferCreateHandleDMA (LPUART_Type ∗ *base,* lpuart_dma_handle_t ∗ *handle,* lpuart_dma_transfer_callback_t *callback,* void ∗ *userData,* dma_handle_t ∗ *txDmaHandle,* dma_handle_t ∗ *rxDmaHandle*)

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_dma_handle_t structure.
callback	Callback function.
userData	User data.
txDmaHandle	User-requested DMA handle for TX DMA transfer.
rxDmaHandle	User-requested DMA handle for RX DMA transfer.

20.3.4.2 status_t LPUART_TransferSendDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, lpuart_transfer_t * *xfer*)

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART DMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_LPUART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

20.3.4.3 status_t LPUART_TransferReceiveDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, lpuart_transfer_t * *xfer*)

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_dma_handle_t structure.
xfer	LPUART DMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_LPUART_RxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

20.3.4.4 void LPUART_TransferAbortSendDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*)

This function aborts send data using DMA.

Parameters

base	LPUART peripheral base address
handle	Pointer to lpuart_dma_handle_t structure

20.3.4.5 void LPUART_TransferAbortReceiveDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*)

This function aborts the received data using DMA.

Parameters

base	LPUART peripheral base address
handle	Pointer to lpuart_dma_handle_t structure

20.3.4.6 status_t LPUART_TransferGetSendCountDMA (LPUART_Type * *base*, lpuart_dma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of bytes that have been written to LPUART TX register by DMA.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferIn-Progress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

20.3.4.7 status_t LPUART_TransferGetReceiveCountDMA (LPUART_Type ∗ *base*, lpuart_dma_handle_t ∗ *handle*, uint32_t ∗ *count*)

This function gets the number of received bytes.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferIn-Progress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

20.4 LPUART eDMA Driver

20.4.1 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.

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.

20.4.2 Data Structure Documentation

20.4.2.1 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.

20.4.2.1.0.61 Field Documentation

20.4.2.1.0.61.1 lpuart_edma_transfer_callback_t lpuart_edma_handle_t::callback

20.4.2.1.0.61.2 void∗ lpuart_edma_handle_t::userData

20.4.2.1.0.61.3 size_t lpuart_edma_handle_t::rxDataSizeAll

20.4.2.1.0.61.4 size_t lpuart_edma_handle_t::txDataSizeAll

20.4.2.1.0.61.5 edma_handle_t∗ lpuart_edma_handle_t::txEdmaHandle

20.4.2.1.0.61.6 edma_handle_t∗ lpuart_edma_handle_t::rxEdmaHandle

20.4.2.1.0.61.7 uint8_t lpuart_edma_handle_t::nbytes

20.4.2.1.0.61.8 volatile uint8_t lpuart_edma_handle_t::txState

20.4.3 Typedef Documentation

20.4.3.1 typedef void(∗ lpuart_edma_transfer_callback_t)(LPUART_Type ∗base, lpuart_edma_handle_t ∗handle, status_t status, void ∗userData)

20.4.4 Function Documentation

20.4.4.1 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*)

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.
callback	Callback function.
userData	User data.
txEdmaHandle	User requested DMA handle for TX DMA transfer.
rxEdmaHandle	User requested DMA handle for RX DMA transfer.

20.4.4.2 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.
xfer	LPUART eDMA transfer structure. See lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others failed.
kStatus_LPUART_TxBusy	Previous transfer on going.
kStatus_InvalidArgument	Invalid argument.

20.4.4.3 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

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.
xfer	LPUART eDMA transfer structure, see lpuart_transfer_t.

Return values

kStatus_Success	if succeed, others fail.
kStatus_LPUART_RxBusy	Previous transfer ongoing.
kStatus_InvalidArgument	Invalid argument.

20.4.4.4 void LPUART_TransferAbortSendEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the sent data using eDMA.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.

20.4.4.5 void LPUART_TransferAbortReceiveEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*)

This function aborts the received data using eDMA.

Parameters

base	LPUART peripheral base address.
handle	Pointer to lpuart_edma_handle_t structure.

20.4.4.6 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

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Send bytes count.

Return values

kStatus_NoTransferIn-Progress	No send in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

20.4.4.7 status_t LPUART_TransferGetReceiveCountEDMA (LPUART_Type * *base*, lpuart_edma_handle_t * *handle*, uint32_t * *count*)

This function gets the number of received bytes.

Parameters

base	LPUART peripheral base address.
handle	LPUART handle pointer.
count	Receive bytes count.

Return values

kStatus_NoTransferIn-Progress	No receive in progress.
kStatus_InvalidArgument	Parameter is invalid.
kStatus_Success	Get successfully through the parameter `count`;

20.5 LPUART FreeRTOS Driver

20.5.1 Overview

Data Structures

- struct lpuart_rtos_config_t
  LPUART RTOS configuration structure. More...

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, const 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.

20.5.2 Data Structure Documentation

20.5.2.1 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.

20.5.3 Function Documentation

20.5.3.1 int LPUART_RTOS_Init (lpuart_rtos_handle_t * *handle,* lpuart_handle_t * *t_handle,* const lpuart_rtos_config_t * *cfg*)

Parameters

handle	The RTOS LPUART handle, the pointer to an allocated space for RTOS context.
t_handle	The pointer to an allocated space to store the transactional layer internal state.
cfg	The pointer to the parameters required to configure the LPUART after initialization.

Returns

0 succeed, others failed

20.5.3.2 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

handle	The RTOS LPUART handle.

20.5.3.3 int LPUART_RTOS_Send (lpuart_rtos_handle_t ∗ *handle,* const 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

handle	The RTOS LPUART handle.
buffer	The pointer to buffer to send.
length	The number of bytes to send.

20.5.3.4 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

handle	The RTOS LPUART handle.
buffer	The pointer to buffer where to write received data.
length	The number of bytes to receive.
received	The pointer to a variable of size_t where the number of received data is filled.

Chapter 21
LTC: LP Trusted Cryptography

21.1 Overview

The MCUXpresso SDK provides the Peripheral driver for the LP Trusted Cryptography (LTC) module of MCUXpresso SDK devices. LP Trusted Cryptography is a set of cryptograhpic hardware accelerator engines that share common registers. LTC architecture can support AES, DES, 3DES, MDHA (SHA), RSA and ECC. Actual list of implemented cryptograhpic hardware accelerator engines depends on specific microcontroller.

The driver comprises two sets of API functions.

In the first set, blocking synchronous APIs are provided, for all operations supported by LTC hardware. The LTC operations are complete (and results are made availabe for further usage) when a function returns. When called, these functions don't return until an LTC operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and also for plaintext or ciphertext data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

In the second set, DMA support for symmetric LTC processing is provided, for AES and DES engines. APIs in the second set use DMA for data movement to and from the LTC input and output FIFOs. By using these functions, main CPU is not used for plaintext or ciphertext data movements (DMA is used instead). Thus, CPU processing power can be used for other application tasks, at cost of decreased maximum data throughput (because of DMA module and transactions management overhead). These functions provide less typical interface, for applications that must offload main CPU while ciphertext or plaintext is being processed, at cost of longer cryptograhpic processing time.

21.2 LTC Driver Initialization and Configuration

LTC Driver is initialized by calling the LTC_Init() function, it enables the LTC module clock in the SIM module. If AES or DES engine is used and the LTC module implementation features the LTC DPA Mask Seed register, seed the DPA mask generator by using the seed from a random number generator. The LTC_SetDpaMaskSeed() function is provided to set the DPA mask seed.

21.3 Comments about API usage in RTOS

LTC operations provided by this driver are not re-entrant. Thus, application software shall ensure the LTC module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

21.4 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).

21.5 LTC Driver Examples

21.5.1 Simple examples

Initialize LTC after Power On Reset or reset cycle

```
LTC_Init(LTC0);
/* optionally initialize DPA mask seed register */
LTC_SetDpaMaskSeed(randomNumber);
```

Encrypt plaintext by DES engine

```
char plain[16];
char cipher[16];

char iv[LTC_DES_IV_SIZE];
char key1[LTC_DES_KEY_SIZE];
char key2[LTC_DES_KEY_SIZE];
char key3[LTC_DES_KEY_SIZE];

memcpy(plain, "Hello World!", 12);
memcpy(iv, "initvect", LTC_DES_IV_SIZE);
memcpy(key1, "mykey1aa", LTC_DES_KEY_SIZE);
memcpy(key2, "mykey2bb", LTC_DES_KEY_SIZE);
memcpy(key3, "mykey3cc", LTC_DES_KEY_SIZE);

LTC_DES3_EncryptCbc(LTC0, plain, cipher, 16, iv, key1, key2, key3);
```

Encrypt plaintext by AES engine

```
char plain[16] = {0};
char cipher[16];
char iv[16] = {0};
char key[16] = {0};

memcpy(plain, "Hello World!", 12);
memcpy(iv, "initvectorinitve", 16);
memcpy(key, "__mykey1aa__^^..", 16);

LTC_AES_EncryptCbc(LTC0, plain, cipher, 16, iv, key, 16);
```

Compute keyed hash by AES engine (CMAC)

```
   char message[] = "Hello World!";
   char key[16] = {0};
   char output[16];
   uint32_t szOutput = 16u;

   memcpy(key, "__mykey1aa__^^..", 16);
LTC_HASH(LTC0, kLTC_Sha256, message, sizeof(message), NULL, 0, output, &szOutput);
```

Compute hash by MDHA engine (SHA-256)

```
char message[] = "Hello World!";
char output[32];
uint32_t szOutput = 32u;

LTC_HASH(LTC0, kLTC_Sha256, message, sizeof(message), NULL, output, &szOutput);
```

Compute modular integer exponentiation

```
status_t status;
const char bigA[] = "112233445566778899aabbccddeeff";
const char bigN[] = "aabbaabbaabb112233445566778899aabbccddeefe";
const char bigE[] = "065537";
char A[256], E[256], N[256], res[256];
uint16_t sizeA, sizeE, sizeN, sizeRes;

/* Note LTC PKHA integer format is least significant byte at lowest address.
 * The _read_string() function converts the input string to LTC PKHA integer format
 * and writes sizeof() the integer to the size variable (sizeA, sizeE, sizeN).
 */
_read_string(A, &sizeA, bigA);
_read_string(E, &sizeE, bigN);
_read_string(N, &sizeN, bigE);

status = LTC_PKHA_ModExp(base, A, sizeA, N, sizeN, E, sizeE, res, &sizeRes,
      kLTC_PKHA_IntegerArith,
                          kLTC_PKHA_NormalValue,
      kLTC_PKHA_TimingEqualized);
```

Compute elliptic curve point multiplication

```
status_t status;
ltc_pkha_ecc_point_t B0, res0;
uint8_t bx, by, resx, resy;
uint8_t E[256];
bool isPointOfInfinity;
uint16_t resultSize, sizeE;

/* Example carried out with 1-byte curve params and point coordinates. */
uint8_t size = 1;
uint8_t aCurveParam = 1;
uint8_t bCurveParam = 0;

bx = 9;
by = 5;

B0.X = &bx;
B0.Y = &by;
res0.X = &resx;
res0.Y = &resy;

/* Prime modulus of the field. */
N[0] = 23;

/* Note LTC PKHA integer has least significant byte at lowest address */

/* Scalar multiplier */
char ew[] = "0100"; /* 256 in decimal */
_read_string(E, &sizeE, ew);

status = LTC_PKHA_ECC_PointMul(LTC0, &B0, E, sizeE, N, NULL, &aCurveParam, &
      bCurveParam, size,
                              kLTC_PKHA_TimingEqualized,
      kLTC_PKHA_IntegerArith, &res0, &isPointOfInfinity);
```

Modules

- LTC Blocking APIs
- LTC Non-blocking eDMA APIs

Functions

- void LTC_Init (LTC_Type ∗base)
  Initializes the LTC driver.
- void LTC_Deinit (LTC_Type ∗base)
  Deinitializes the LTC driver.

Driver version

- #define FSL_LTC_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))
  LTC driver version.

21.6 Macro Definition Documentation

21.6.1 #define FSL_LTC_DRIVER_VERSION (MAKE_VERSION(2, 0, 4))

Version 2.0.4.

Current version: 2.0.4

Change log:

- Version 2.0.1
  - fixed warning during g++ compilation
- Version 2.0.2
  - fixed [KPSDK-10932][LTC][SHA] LTC_HASH() blocks indefinitely when message size exceeds 4080 bytes
- Version 2.0.3
  - fixed LTC_PKHA_CompareBigNum() in case an integer argument is an array of all zeroes
- Version 2.0.4
  - constant LTC_PKHA_CompareBigNum() processing time

21.7 Function Documentation

21.7.1 void LTC_Init (LTC_Type ∗ *base*)

This function initializes the LTC driver.

Parameters

base	LTC peripheral base address

21.7.2 void LTC_Deinit (LTC_Type ∗ *base*)

This function deinitializes the LTC driver.

Parameters

base	LTC peripheral base address

21.8 LTC Blocking APIs

21.8.1 Overview

This section describes the programming interface of the LTC Synchronous Blocking functions

Modules

- LTC AES driver
- LTC DES driver
- LTC HASH driver
- LTC PKHA driver

21.8.2 LTC DES driver

21.8.2.1 Overview

This section describes the programming interface of the LTC DES driver.

Macros

- #define LTC_DES_KEY_SIZE 8
  LTC DES key size - 64 bits.
- #define LTC_DES_IV_SIZE 8
  LTC DES IV size - 8 bytes.

Functions

- status_t LTC_DES_EncryptEcb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using ECB block mode.
- status_t LTC_DES_DecryptEcb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using ECB block mode.
- status_t LTC_DES_EncryptCbc (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using CBC block mode.
- status_t LTC_DES_DecryptCbc (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using CBC block mode.
- status_t LTC_DES_EncryptCfb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using CFB block mode.
- status_t LTC_DES_DecryptCfb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using CFB block mode.
- status_t LTC_DES_EncryptOfb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using OFB block mode.
- status_t LTC_DES_DecryptOfb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using OFB block mode.
- status_t LTC_DES2_EncryptEcb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Encrypts triple DES using ECB block mode with two keys.
- status_t LTC_DES2_DecryptEcb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Decrypts triple DES using ECB block mode with two keys.

- status_t LTC_DES2_EncryptCbc (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Encrypts triple DES using CBC block mode with two keys.
- status_t LTC_DES2_DecryptCbc (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Decrypts triple DES using CBC block mode with two keys.
- status_t LTC_DES2_EncryptCfb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Encrypts triple DES using CFB block mode with two keys.
- status_t LTC_DES2_DecryptCfb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Decrypts triple DES using CFB block mode with two keys.
- status_t LTC_DES2_EncryptOfb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Encrypts triple DES using OFB block mode with two keys.
- status_t LTC_DES2_DecryptOfb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Decrypts triple DES using OFB block mode with two keys.
- status_t LTC_DES3_EncryptEcb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Encrypts triple DES using ECB block mode with three keys.
- status_t LTC_DES3_DecryptEcb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Decrypts triple DES using ECB block mode with three keys.
- status_t LTC_DES3_EncryptCbc (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Encrypts triple DES using CBC block mode with three keys.
- status_t LTC_DES3_DecryptCbc (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Decrypts triple DES using CBC block mode with three keys.
- status_t LTC_DES3_EncryptCfb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Encrypts triple DES using CFB block mode with three keys.
- status_t LTC_DES3_DecryptCfb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Decrypts triple DES using CFB block mode with three keys.

- status_t LTC_DES3_EncryptOfb (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Encrypts triple DES using OFB block mode with three keys.
- status_t LTC_DES3_DecryptOfb (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])
  Decrypts triple DES using OFB block mode with three keys.

21.8.2.2 Macro Definition Documentation

21.8.2.2.1 #define LTC_DES_KEY_SIZE 8

21.8.2.3 Function Documentation

21.8.2.3.1 status_t LTC_DES_EncryptEcb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using ECB block mode.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
`out`	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.2 status_t LTC_DES_DecryptEcb (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using ECB block mode.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.3 status_t LTC_DES_EncryptCbc (LTC_Type ∗ *base,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using CBC block mode.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Ouput ciphertext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.4 status_t LTC_DES_DecryptCbc (LTC_Type ∗ *base,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using CBC block mode.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.5 status_t LTC_DES_EncryptCfb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using CFB block mode.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
	size	Size of input data in bytes
	iv	Input initial block.
	key	Input key to use for encryption
out	*ciphertext*	Output ciphertext

Returns

Status from encrypt/decrypt operation

21.8.2.3.6 status_t LTC_DES_DecryptCfb (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using CFB block mode.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial block.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.7 status_t LTC_DES_EncryptOfb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using OFB block mode.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.8 status_t LTC_DES_DecryptOfb (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using OFB block mode.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.8.2.3.9 status_t LTC_DES2_EncryptEcb (LTC_Type ∗ *base,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using ECB block mode with two keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.10 status_t LTC_DES2_DecryptEcb (LTC_Type ∗ *base,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using ECB block mode with two keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.11 status_t LTC_DES2_EncryptCbc (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CBC block mode with two keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.12 status_t LTC_DES2_DecryptCbc (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CBC block mode with two keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.13 status_t LTC_DES2_EncryptCfb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.14 status_t LTC_DES2_DecryptCfb (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.15 status_t LTC_DES2_EncryptOfb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using OFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.16 status_t LTC_DES2_DecryptOfb (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using OFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.17 status_t LTC_DES3_EncryptEcb (LTC_Type ∗ *base*, const uint8_t ∗ *plaintext*, uint8_t ∗ *ciphertext*, uint32_t *size*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using ECB block mode with three keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.18 status_t LTC_DES3_DecryptEcb (LTC_Type ∗ *base*, const uint8_t ∗ *ciphertext*, uint8_t ∗ *plaintext*, uint32_t *size*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using ECB block mode with three keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.19 status_t LTC_DES3_EncryptCbc (LTC_Type ∗ *base*, const uint8_t ∗ *plaintext*, uint8_t ∗ *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CBC block mode with three keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.20 status_t LTC_DES3_DecryptCbc (LTC_Type * *base,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CBC block mode with three keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.21 status_t LTC_DES3_EncryptCfb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and ouput data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.22 status_t LTC_DES3_DecryptCfb (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.23 status_t LTC_DES3_EncryptOfb (LTC_Type ∗ *base,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using OFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.2.3.24 status_t LTC_DES3_DecryptOfb (LTC_Type * *base,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using OFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	ciphertext	Input ciphertext to decrypt
`out`	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.8.3 LTC AES driver

21.8.3.1 Overview

This section describes the programming interface of the LTC AES driver.

Macros

- #define LTC_AES_BLOCK_SIZE 16
  AES block size in bytes.
- #define LTC_AES_IV_SIZE 16
  AES Input Vector size in bytes.
- #define LTC_AES_DecryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft) LTC_AES_CryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)
  AES CTR decrypt is mapped to the AES CTR generic operation.
- #define LTC_AES_EncryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft) LTC_AES_CryptCtr(base, input, output, size, counter, key, keySize, counterlast, szLeft)
  AES CTR encrypt is mapped to the AES CTR generic operation.

Enumerations

- enum ltc_aes_key_t {
  kLTC_EncryptKey = 0U,
  kLTC_DecryptKey = 1U }
  Type of AES key for ECB and CBC decrypt operations.

Functions

- status_t LTC_AES_GenerateDecryptKey (LTC_Type ∗base, const uint8_t ∗encryptKey, uint8_t ∗decryptKey, uint32_t keySize)
  Transforms an AES encrypt key (forward AES) into the decrypt key (inverse AES).
- status_t LTC_AES_EncryptEcb (LTC_Type ∗base, const uint8_t ∗plaintext, uint8_t ∗ciphertext, uint32_t size, const uint8_t ∗key, uint32_t keySize)
  Encrypts AES using the ECB block mode.
- status_t LTC_AES_DecryptEcb (LTC_Type ∗base, const uint8_t ∗ciphertext, uint8_t ∗plaintext, uint32_t size, const uint8_t ∗key, uint32_t keySize, ltc_aes_key_t keyType)
  Decrypts AES using ECB block mode.
- status_t LTC_AES_EncryptCbc (LTC_Type ∗base, const uint8_t ∗plaintext, uint8_t ∗ciphertext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t ∗key, uint32_t keySize)
  Encrypts AES using CBC block mode.
- status_t LTC_AES_DecryptCbc (LTC_Type ∗base, const uint8_t ∗ciphertext, uint8_t ∗plaintext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t ∗key, uint32_t keySize, ltc_aes_key_t keyType)
  Decrypts AES using CBC block mode.
- status_t LTC_AES_CryptCtr (LTC_Type ∗base, const uint8_t ∗input, uint8_t ∗output, uint32_t size, uint8_t counter[LTC_AES_BLOCK_SIZE], const uint8_t ∗key, uint32_t keySize, uint8_t

counterlast[LTC_AES_BLOCK_SIZE], uint32_t *szLeft)
Encrypts or decrypts AES using CTR block mode.

- status_t LTC_AES_EncryptTagCcm (LTC_Type *base, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, uint8_t *tag, uint32_t tagSize)
  Encrypts AES and tags using CCM block mode.
- status_t LTC_AES_DecryptTagCcm (LTC_Type *base, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t *iv, uint32_t ivSize, const uint8_t *aad, uint32_t aadSize, const uint8_t *key, uint32_t keySize, const uint8_t *tag, uint32_t tagSize)
  Decrypts AES and authenticates using CCM block mode.

21.8.3.2 Enumeration Type Documentation

21.8.3.2.1 enum ltc_aes_key_t

Enumerator

kLTC_EncryptKey Input key is an encrypt key.
kLTC_DecryptKey Input key is a decrypt key.

21.8.3.3 Function Documentation

21.8.3.3.1 status_t LTC_AES_GenerateDecryptKey (LTC_Type * *base*, const uint8_t * *encryptKey*, uint8_t * *decryptKey*, uint32_t *keySize*)

Transforms the AES encrypt key (forward AES) into the decrypt key (inverse AES). The key derived by this function can be used as a direct load decrypt key for AES ECB and CBC decryption operations (keyType argument).

Parameters

	base	LTC peripheral base address
	encryptKey	Input key for decrypt key transformation
out	*decryptKey*	Output key, the decrypt form of the AES key.
	keySize	Size of the input key and output key in bytes. Must be 16, 24, or 32.

Returns

Status from key generation operation

21.8.3.3.2 status_t LTC_AES_EncryptEcb (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t * *key*, uint32_t *keySize*)

Encrypts AES using the ECB block mode.

Parameters

	base	LTC peripheral base address
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	key	Input key to use for encryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

21.8.3.3.3 status_t LTC_AES_DecryptEcb (LTC_Type ∗ *base,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t ∗ *key,* uint32_t *keySize,* ltc_aes_key_t *keyType*)

Decrypts AES using ECB block mode.

Parameters

	base	LTC peripheral base address
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	key	Input key.
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
	keyType	Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

21.8.3.3.4 status_t LTC_AES_EncryptCbc (LTC_Type ∗ *base,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_AES_IV_SIZE],* const uint8_t ∗ *key,* uint32_t *keySize*)

Parameters

	base	LTC peripheral base address
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.
	key	Input key to use for encryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

21.8.3.3.5 status_t LTC_AES_DecryptCbc (LTC_Type * *base*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_AES_IV_SIZE]*, const uint8_t * *key*, uint32_t *keySize*, ltc_aes_key_t *keyType*)

Parameters

	base	LTC peripheral base address
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.
	key	Input key to use for decryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
	keyType	Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

Returns

Status from decrypt operation

21.8.3.3.6 status_t LTC_AES_CryptCtr (LTC_Type * *base*, const uint8_t * *input*, uint8_t * *output*, uint32_t *size*, uint8_t *counter[LTC_AES_BLOCK_SIZE]*, const uint8_t * *key*, uint32_t *keySize*, uint8_t *counterlast[LTC_AES_BLOCK_SIZE]*, uint32_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

	base	LTC peripheral base address
	input	Input data for CTR block mode
`out`	*output*	Output data for CTR block mode
	size	Size of input and output data in bytes
`in,out`	*counter*	Input counter (updates on return)
	key	Input key to use for forward AES cipher
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
`out`	*counterlast*	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
`out`	*szLeft*	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

21.8.3.3.7 status_t LTC_AES_EncryptTagCcm (LTC_Type * *base*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t * *iv*, uint32_t *ivSize*, const uint8_t * *aad*, uint32_t *aadSize*, const uint8_t * *key*, uint32_t *keySize*, uint8_t * *tag*, uint32_t *tagSize*)

Encrypts AES and optionally tags using CCM block mode.

Parameters

	base	LTC peripheral base address
	plaintext	Input plain text to encrypt
`out`	*ciphertext*	Output cipher text.
	size	Size of input and output data in bytes. Zero means authentication only.
	iv	Nonce

	ivSize	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	aad	Input additional authentication data. Can be NULL if aadSize is zero.
	aadSize	Input size in bytes of AAD. Zero means data mode only (authentication skipped).
	key	Input key to use for encryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
`out`	*tag*	Generated output tag. Set to NULL to skip tag processing.
	tagSize	Input size of the tag to generate, in bytes. Must be 4, 6, 8, 10, 12, 14, or 16.

Returns

Status from encrypt operation

21.8.3.3.8 status_t LTC_AES_DecryptTagCcm (LTC_Type ∗ *base,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t ∗ *iv,* uint32_t *ivSize,* const uint8_t ∗ *aad,* uint32_t *aadSize,* const uint8_t ∗ *key,* uint32_t *keySize,* const uint8_t ∗ *tag,* uint32_t *tagSize*)

Decrypts AES and optionally authenticates using CCM block mode.

Parameters

	base	LTC peripheral base address
	ciphertext	Input cipher text to decrypt
`out`	*plaintext*	Output plain text.
	size	Size of input and output data in bytes. Zero means authentication only.
	iv	Nonce
	ivSize	Length of the Nonce in bytes. Must be 7, 8, 9, 10, 11, 12, or 13.
	aad	Input additional authentication data. Can be NULL if aadSize is zero.
	aadSize	Input size in bytes of AAD. Zero means data mode only (authentication skipped).

	key	Input key to use for decryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
	tag	Received tag. Set to NULL to skip tag processing.
	tagSize	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

21.8.4 LTC HASH driver

21.8.4.1 Overview

This section describes the programming interface of the LTC HASH driver.

Macros

- #define LTC_HASH_CTX_SIZE 29
  LTC HASH Context size.

Typedefs

- typedef uint32_t ltc_hash_ctx_t [LTC_HASH_CTX_SIZE]
  Storage type used to save hash context.

Enumerations

- enum ltc_hash_algo_t {
  kLTC_XcbcMac = 0,
  kLTC_Cmac }
  Supported cryptographic block cipher functions for HASH creation.

Functions

- status_t LTC_HASH_Init (LTC_Type *base, ltc_hash_ctx_t *ctx, ltc_hash_algo_t algo, const uint8_t *key, uint32_t keySize)
  Initialize HASH context.
- status_t LTC_HASH_Update (ltc_hash_ctx_t *ctx, const uint8_t *input, uint32_t inputSize)
  Add data to current HASH.
- status_t LTC_HASH_Finish (ltc_hash_ctx_t *ctx, uint8_t *output, uint32_t *outputSize)
  Finalize hashing.
- status_t LTC_HASH (LTC_Type *base, ltc_hash_algo_t algo, const uint8_t *input, uint32_t inputSize, const uint8_t *key, uint32_t keySize, uint8_t *output, uint32_t *outputSize)
  Create HASH on given data.

21.8.4.2 Macro Definition Documentation

21.8.4.2.1 #define LTC_HASH_CTX_SIZE 29

21.8.4.3 Typedef Documentation

21.8.4.3.1 typedef uint32_t ltc_hash_ctx_t[LTC_HASH_CTX_SIZE]

21.8.4.4 Enumeration Type Documentation

21.8.4.4.1 enum ltc_hash_algo_t

Enumerator

kLTC_XcbcMac XCBC-MAC (AES engine)
kLTC_Cmac CMAC (AES engine)

21.8.4.5 Function Documentation

21.8.4.5.1 status_t LTC_HASH_Init (LTC_Type * *base,* ltc_hash_ctx_t * *ctx,* ltc_hash_algo_t *algo,* const uint8_t * *key,* uint32_t *keySize*)

This function initialize the HASH. Key shall be supplied if the underlaying algoritm is AES XCBC-MAC or CMAC. Key shall be NULL if the underlaying algoritm 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.

Parameters

	base	LTC peripheral base address
out	*ctx*	Output hash context
	algo	Underlaying algorithm to use for hash computation.
	key	Input key (NULL if underlaying algorithm is SHA)
	keySize	Size of input key in bytes

Returns

Status of initialization

21.8.4.5.2 status_t LTC_HASH_Update (ltc_hash_ctx_t * *ctx,* const uint8_t * *input,* uint32_t *inputSize*)

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

Parameters

in,out	*ctx*	HASH context
	input	Input data
	inputSize	Size of input data in bytes

Returns

Status of the hash update operation

21.8.4.5.3 status_t LTC_HASH_Finish (ltc_hash_ctx_t * *ctx*, uint8_t * *output*, uint32_t * *outputSize*)

Outputs the final hash and erases the context.

Parameters

in,out	*ctx*	Input hash context
out	*output*	Output hash data
out	*outputSize*	Output parameter storing the size of the output hash in bytes

Returns

Status of the hash finish operation

21.8.4.5.4 status_t LTC_HASH (LTC_Type * *base*, ltc_hash_algo_t *algo*, const uint8_t * *input*, uint32_t *inputSize*, const uint8_t * *key*, uint32_t *keySize*, uint8_t * *output*, uint32_t * *outputSize*)

Perform the full keyed HASH in one function call.

Parameters

	base	LTC peripheral base address
	algo	Block cipher algorithm to use for CMAC creation

	input	Input data
	inputSize	Size of input data in bytes
	key	Input key
	keySize	Size of input key in bytes
`out`	*output*	Output hash data
`out`	*outputSize*	Output parameter storing the size of the output hash in bytes

Returns

Status of the one call hash operation.

21.8.5 LTC PKHA driver

21.8.5.1 Overview

This section describes the programming interface of the LTC PKHA driver.

Data Structures

- struct ltc_pkha_ecc_point_t
  PKHA ECC point structure. More...

Enumerations

- enum ltc_pkha_timing_t {
  kLTC_PKHA_NoTimingEqualized = 0U,
  kLTC_PKHA_TimingEqualized = 1U }
  Use of timing equalized version of a PKHA function.
- enum ltc_pkha_f2m_t {
  kLTC_PKHA_IntegerArith = 0U,
  kLTC_PKHA_F2mArith = 1U }
  Integer vs binary polynomial arithmetic selection.
- enum ltc_pkha_montgomery_form_t {
  kLTC_PKHA_NormalValue = 0U,
  kLTC_PKHA_MontgomeryFormat = 1U }
  Montgomery or normal PKHA input format.

Functions

- int LTC_PKHA_CompareBigNum (const uint8_t *a, size_t sizeA, const uint8_t *b, size_t sizeB)
  Compare two PKHA big numbers.
- status_t LTC_PKHA_NormalToMontgomery (LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, uint8_t *R2, uint16_t *sizeR2, ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t arithType)
  Converts from integer to Montgomery format.
- status_t LTC_PKHA_MontgomeryToNormal (LTC_Type *base, const uint8_t *N, uint16_t sizeN, uint8_t *A, uint16_t *sizeA, uint8_t *B, uint16_t *sizeB, ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t arithType)
  Converts from Montgomery format to int.
- status_t LTC_PKHA_ModAdd (LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize, ltc_pkha_f2m_t arithType)
  Performs modular addition - (A + B) mod N.
- status_t LTC_PKHA_ModSub1 (LTC_Type *base, const uint8_t *A, uint16_t sizeA, const uint8_t *B, uint16_t sizeB, const uint8_t *N, uint16_t sizeN, uint8_t *result, uint16_t *resultSize)
  Performs modular subtraction - (A - B) mod N.

- status_t LTC_PKHA_ModSub2 (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗B, uint16_t sizeB, const uint8_t ∗N, uint16_t sizeN, uint8_t ∗result, uint16_t ∗resultSize)
  Performs modular subtraction - (B - A) mod N.
- status_t LTC_PKHA_ModMul (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗B, uint16_t sizeB, const uint8_t ∗N, uint16_t sizeN, uint8_t ∗result, uint16_t ∗resultSize, ltc_pkha_f2m_t arithType, ltc_pkha_montgomery_form_t montIn, ltc_pkha_montgomery_form_t montOut, ltc_pkha_timing_t equalTime)
  Performs modular multiplication - (A x B) mod N.
- status_t LTC_PKHA_ModExp (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗N, uint16_t sizeN, const uint8_t ∗E, uint16_t sizeE, uint8_t ∗result, uint16_t ∗resultSize, ltc_pkha_f2m_t arithType, ltc_pkha_montgomery_form_t montIn, ltc_pkha_timing_t equalTime)
  Performs modular exponentiation - (A^E) mod N.
- status_t LTC_PKHA_ModRed (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗N, uint16_t sizeN, uint8_t ∗result, uint16_t ∗resultSize, ltc_pkha_f2m_t arithType)
  Performs modular reduction - (A) mod N.
- status_t LTC_PKHA_ModInv (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗N, uint16_t sizeN, uint8_t ∗result, uint16_t ∗resultSize, ltc_pkha_f2m_t arithType)
  Performs modular inversion - (A^-1) mod N.
- status_t LTC_PKHA_ModR2 (LTC_Type ∗base, const uint8_t ∗N, uint16_t sizeN, uint8_t ∗result, uint16_t ∗resultSize, ltc_pkha_f2m_t arithType)
  Computes integer Montgomery factor R^2 mod N.
- status_t LTC_PKHA_GCD (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗N, uint16_t sizeN, uint8_t ∗result, uint16_t ∗resultSize, ltc_pkha_f2m_t arithType)
  Calculates the greatest common divisor - GCD (A, N).
- status_t LTC_PKHA_PrimalityTest (LTC_Type ∗base, const uint8_t ∗A, uint16_t sizeA, const uint8_t ∗B, uint16_t sizeB, const uint8_t ∗N, uint16_t sizeN, bool ∗res)
  Executes Miller-Rabin primality test.
- status_t LTC_PKHA_ECC_PointAdd (LTC_Type ∗base, const ltc_pkha_ecc_point_t ∗A, const ltc_pkha_ecc_point_t ∗B, const uint8_t ∗N, const uint8_t ∗R2modN, const uint8_t ∗aCurveParam, const uint8_t ∗bCurveParam, uint8_t size, ltc_pkha_f2m_t arithType, ltc_pkha_ecc_point_t ∗result)
  Adds elliptic curve points - A + B.
- status_t LTC_PKHA_ECC_PointDouble (LTC_Type ∗base, const ltc_pkha_ecc_point_t ∗B, const uint8_t ∗N, const uint8_t ∗aCurveParam, const uint8_t ∗bCurveParam, uint8_t size, ltc_pkha_f2m_t arithType, ltc_pkha_ecc_point_t ∗result)
  Doubles elliptic curve points - B + B.
- status_t LTC_PKHA_ECC_PointMul (LTC_Type ∗base, const ltc_pkha_ecc_point_t ∗A, const uint8_t ∗E, uint8_t sizeE, const uint8_t ∗N, const uint8_t ∗R2modN, const uint8_t ∗aCurveParam, const uint8_t ∗bCurveParam, uint8_t size, ltc_pkha_timing_t equalTime, ltc_pkha_f2m_t arithType, ltc_pkha_ecc_point_t ∗result, bool ∗infinity)
  Multiplies an elliptic curve point by a scalar - E x (A0, A1).

21.8.5.2 Data Structure Documentation

21.8.5.2.1 struct ltc_pkha_ecc_point_t

Data Fields

- uint8_t ∗ X
  X coordinate (affine)
- uint8_t ∗ Y
  Y coordinate (affine)

21.8.5.3 Enumeration Type Documentation

21.8.5.3.1 enum ltc_pkha_timing_t

Enumerator

kLTC_PKHA_NoTimingEqualized Normal version of a PKHA operation.
kLTC_PKHA_TimingEqualized Timing-equalized version of a PKHA operation.

21.8.5.3.2 enum ltc_pkha_f2m_t

Enumerator

kLTC_PKHA_IntegerArith Use integer arithmetic.
kLTC_PKHA_F2mArith Use binary polynomial arithmetic.

21.8.5.3.3 enum ltc_pkha_montgomery_form_t

Enumerator

kLTC_PKHA_NormalValue PKHA number is normal integer.
kLTC_PKHA_MontgomeryFormat PKHA number is in montgomery format.

21.8.5.4 Function Documentation

21.8.5.4.1 int LTC_PKHA_CompareBigNum (const uint8_t ∗ *a*, size_t *sizeA*, const uint8_t ∗ *b*, size_t *sizeB*)

Compare two PKHA big numbers. Return 1 for a > b, -1 for a < b and 0 if they are same. PKHA big number is lsbyte first. Thus the comparison starts at msbyte which is the last member of tested arrays.

Parameters

a	First integer represented as an array of bytes, lsbyte first.
sizeA	Size in bytes of the first integer.
b	Second integer represented as an array of bytes, lsbyte first.
sizeB	Size in bytes of the second integer.

Returns

1 if a > b.
-1 if a < b.
0 if a = b.

21.8.5.4.2 status_t LTC_PKHA_NormalToMontgomery (LTC_Type * *base*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *A*, uint16_t * *sizeA*, uint8_t * *B*, uint16_t * *sizeB*, uint8_t * *R2*, uint16_t * *sizeR2*, ltc_pkha_timing_t *equalTime*, ltc_pkha_f2m_t *arithType*)

This function computes R2 mod N and optionally converts A or B into Montgomery format of A or B.

Parameters

	base	LTC peripheral base address
	N	modulus
	sizeN	size of N in bytes
in,out	*A*	The first input in non-Montgomery format. Output Montgomery format of the first input.
in,out	*sizeA*	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	*B*	Second input in non-Montgomery format. Output Montgomery format of the second input.
in,out	*sizeB*	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	*R2*	Output Montgomery factor R2 mod N.
out	*sizeR2*	pointer to size variable. On output it holds size of Montgomery factor R2 mod N in bytes.

	equalTime	Run the function time equalized or no timing equalization.
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.3 status_t LTC_PKHA_MontgomeryToNormal (LTC_Type * *base*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *A*, uint16_t * *sizeA*, uint8_t * *B*, uint16_t * *sizeB*, ltc_pkha_timing_t *equalTime*, ltc_pkha_f2m_t *arithType*)

This function converts Montgomery format of A or B into int A or B.

Parameters

	base	LTC peripheral base address
	N	modulus.
	sizeN	size of N modulus in bytes.
`in,out`	*A*	Input first number in Montgomery format. Output is non-Montgomery format.
`in,out`	*sizeA*	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.
`in,out`	*B*	Input first number in Montgomery format. Output is non-Montgomery format.
`in,out`	*sizeB*	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.
	equalTime	Run the function time equalized or no timing equalization.
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.4 status_t LTC_PKHA_ModAdd (LTC_Type * *base*, const uint8_t * *A*, uint16_t *sizeA*, const uint8_t * *B*, uint16_t *sizeB*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*, ltc_pkha_f2m_t *arithType*)

This function performs modular addition of (A + B) mod 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

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes
	B	second addend (integer or binary polynomial)
	sizeB	Size of B in bytes
	N	modulus. For F2m operation this can be NULL, as N is ignored during F2m polynomial addition.
	sizeN	Size of N in bytes. This must be given for both integer and F2m polynomial additions.
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.5 status_t LTC_PKHA_ModSub1 (LTC_Type ∗ *base,* const uint8_t ∗ *A,* uint16_t *sizeA,* const uint8_t ∗ *B,* uint16_t *sizeB,* const uint8_t ∗ *N,* uint16_t *sizeN,* uint8_t ∗ *result,* uint16_t ∗ *resultSize*)

This function performs modular subtraction of (A - B) mod N with integer inputs.

Parameters

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes
	B	second addend (integer or binary polynomial)
	sizeB	Size of B in bytes
	N	modulus

	sizeN	Size of N in bytes
out	*result*	Output array to store result of operation
out	*resultSize*	Output size of operation in bytes

Returns

Operation status.

21.8.5.4.6 status_t LTC_PKHA_ModSub2 (LTC_Type * *base*, const uint8_t * *A*, uint16_t *sizeA*, const uint8_t * *B*, uint16_t *sizeB*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*)

This function performs modular subtraction of (B - A) mod N, with integer inputs.

Parameters

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes
	B	second addend (integer or binary polynomial)
	sizeB	Size of B in bytes
	N	modulus
	sizeN	Size of N in bytes
out	*result*	Output array to store result of operation
out	*resultSize*	Output size of operation in bytes

Returns

Operation status.

21.8.5.4.7 status_t LTC_PKHA_ModMul (LTC_Type * *base*, const uint8_t * *A*, uint16_t *sizeA*, const uint8_t * *B*, uint16_t *sizeB*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*, ltc_pkha_f2m_t *arithType*, ltc_pkha_montgomery_form_t *montIn*, ltc_pkha_montgomery_form_t *montOut*, ltc_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

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes
	B	second addend (integer or binary polynomial)
	sizeB	Size of B in bytes
	N	modulus.
	sizeN	Size of N in bytes
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	arithType	Type of arithmetic to perform (integer or F2m)
	montIn	Format of inputs
	montOut	Format of output
	equalTime	Run the function time equalized or no timing equalization. This argument is ignored for F2m modular multiplication.

Returns

Operation status.

21.8.5.4.8 status_t LTC_PKHA_ModExp (LTC_Type ∗ *base*, const uint8_t ∗ *A*, uint16_t *sizeA*, const uint8_t ∗ *N*, uint16_t *sizeN*, const uint8_t ∗ *E*, uint16_t *sizeE*, uint8_t ∗ *result*, uint16_t ∗ *resultSize*, ltc_pkha_f2m_t *arithType*, ltc_pkha_montgomery_form_t *montIn*, ltc_pkha_timing_t *equalTime*)

This function performs modular exponentiation with either integer or binary polynomial (F2m) inputs.

Parameters

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes

	N	modulus
	sizeN	Size of N in bytes
	E	exponent
	sizeE	Size of E in bytes
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	montIn	Format of A input (normal or Montgomery)
	arithType	Type of arithmetic to perform (integer or F2m)
	equalTime	Run the function time equalized or no timing equalization.

Returns

Operation status.

21.8.5.4.9 status_t LTC_PKHA_ModRed (LTC_Type * *base*, const uint8_t * *A*, uint16_t *sizeA*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*, ltc_pkha_f2m_t *arithType*)

This function performs modular reduction with either integer or binary polynomial (F2m) inputs.

Parameters

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes
	N	modulus
	sizeN	Size of N in bytes
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.10 status_t LTC_PKHA_ModInv (LTC_Type * *base*, const uint8_t * *A*, uint16_t *sizeA*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*, ltc_pkha_f2m_t *arithType*)

This function performs modular inversion with either integer or binary polynomial (F2m) inputs.

Parameters

	base	LTC peripheral base address
	A	first addend (integer or binary polynomial)
	sizeA	Size of A in bytes
	N	modulus
	sizeN	Size of N in bytes
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.11 status_t LTC_PKHA_ModR2 (LTC_Type * *base*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*, ltc_pkha_f2m_t *arithType*)

This function computes a constant to assist in converting operands into the Montgomery residue system representation.

Parameters

	base	LTC peripheral base address
	N	modulus
	sizeN	Size of N in bytes
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.12 status_t LTC_PKHA_GCD (LTC_Type * *base*, const uint8_t * *A*, uint16_t *sizeA*, const uint8_t * *N*, uint16_t *sizeN*, uint8_t * *result*, uint16_t * *resultSize*, ltc_pkha_f2m_t *arithType*)

This function calculates the greatest common divisor of two inputs with either integer or binary polynomial (F2m) inputs.

Parameters

	base	LTC peripheral base address
	A	first value (must be smaller than or equal to N)
	sizeA	Size of A in bytes
	N	second value (must be non-zero)
	sizeN	Size of N in bytes
`out`	*result*	Output array to store result of operation
`out`	*resultSize*	Output size of operation in bytes
	arithType	Type of arithmetic to perform (integer or F2m)

Returns

Operation status.

21.8.5.4.13 status_t LTC_PKHA_PrimalityTest (LTC_Type ∗ *base*, const uint8_t ∗ *A*, uint16_t *sizeA*, const uint8_t ∗ *B*, uint16_t *sizeB*, const uint8_t ∗ *N*, uint16_t *sizeN*, bool ∗ *res*)

This function calculates whether or not a candidate prime number is likely to be a prime.

Parameters

	base	LTC peripheral base address
	A	initial random seed
	sizeA	Size of A in bytes
	B	number of trial runs
	sizeB	Size of B in bytes
	N	candidate prime integer
	sizeN	Size of N in bytes
`out`	*res*	True if the value is likely prime or false otherwise

Returns

Operation status.

21.8.5.4.14 status_t LTC_PKHA_ECC_PointAdd (LTC_Type * *base,* const ltc_pkha_ecc_point_t * *A,* const ltc_pkha_ecc_point_t * *B,* const uint8_t * *N,* const uint8_t * *R2modN,* const uint8_t * *aCurveParam,* const uint8_t * *bCurveParam,* uint8_t *size,* ltc_pkha_f2m_t *arithType,* ltc_pkha_ecc_point_t * *result*)

This function performs ECC point addition over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

	base	LTC peripheral base address
	A	Left-hand point
	B	Right-hand point
	N	Prime modulus of the field
	R2modN	NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from LTC_PKHA_ModR2() function).
	aCurveParam	A parameter from curve equation
	bCurveParam	B parameter from curve equation (constant)
	size	Size in bytes of curve points and parameters
	arithType	Type of arithmetic to perform (integer or F2m)
`out`	*result*	Result point

Returns

Operation status.

21.8.5.4.15 status_t LTC_PKHA_ECC_PointDouble (LTC_Type * *base*, const ltc_pkha_ecc_point_t * *B*, const uint8_t * *N*, const uint8_t * *aCurveParam*, const uint8_t * *bCurveParam*, uint8_t *size*, ltc_pkha_f2m_t *arithType*, ltc_pkha_ecc_point_t * *result*)

This function performs ECC point doubling over a prime field (Fp) or binary field (F2m) using affine coordinates.

Parameters

	base	LTC peripheral base address
	B	Point to double
	N	Prime modulus of the field
	aCurveParam	A parameter from curve equation
	bCurveParam	B parameter from curve equation (constant)

	size	Size in bytes of curve points and parameters
	arithType	Type of arithmetic to perform (integer or F2m)
out	*result*	Result point

Returns

Operation status.

21.8.5.4.16 status_t LTC_PKHA_ECC_PointMul (LTC_Type * *base*, const ltc_pkha_ecc_point_t * *A*, const uint8_t * *E*, uint8_t *sizeE*, const uint8_t * *N*, const uint8_t * *R2modN*, const uint8_t * *aCurveParam*, const uint8_t * *bCurveParam*, uint8_t *size*, ltc_pkha_timing_t *equalTime*, ltc_pkha_f2m_t *arithType*, ltc_pkha_ecc_point_t * *result*, bool * *infinity*)

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

	base	LTC peripheral base address
	A	Point as multiplicand
	E	Scalar multiple
	sizeE	The size of E, in bytes
	N	Modulus, a prime number for the Fp field or Irreducible polynomial for F2m field.
	R2modN	NULL (the function computes R2modN internally) or pointer to pre-computed R2modN (obtained from LTC_PKHA_ModR2() function).
	aCurveParam	A parameter from curve equation
	bCurveParam	B parameter from curve equation (C parameter for operation over F2m).
	size	Size in bytes of curve points and parameters
	equalTime	Run the function time equalized or no timing equalization.
	arithType	Type of arithmetic to perform (integer or F2m)

`out`	*result*	Result point
`out`	*infinity*	Output true if the result is point of infinity, and false otherwise. Writing of this output will be ignored if the argument is NULL.

Returns

Operation status.

21.9 LTC Non-blocking eDMA APIs

21.9.1 Overview

This section describes the programming interface of the LTC eDMA Non Blocking functions

Modules

- LTC eDMA AES driver
- LTC eDMA DES driver

Data Structures

- struct ltc_edma_handle_t
  LTC eDMA handle. More...

Typedefs

- typedef void(∗ ltc_edma_callback_t)(LTC_Type ∗base, ltc_edma_handle_t ∗handle, status_t status, void ∗userData)
  LTC eDMA callback function.
- typedef status_t(∗ ltc_edma_state_machine_t)(LTC_Type ∗base, ltc_edma_handle_t ∗handle)
  LTC eDMA state machine function.

Functions

- void LTC_CreateHandleEDMA (LTC_Type ∗base, ltc_edma_handle_t ∗handle, ltc_edma_callback_t callback, void ∗userData, edma_handle_t ∗inputFifoEdmaHandle, edma_handle_t ∗outputFifoEdmaHandle)
  Init the LTC eDMA handle which is used in transcational functions.

21.9.2 Data Structure Documentation

21.9.2.1 struct _ltc_edma_handle

It is defined only for private usage inside LTC eDMA driver.

Data Fields

- ltc_edma_callback_t callback
  Callback function.
- void ∗ userData

LTC callback function parameter.

- edma_handle_t * inputFifoEdmaHandle

  The eDMA TX channel used.
- edma_handle_t * outputFifoEdmaHandle

  The eDMA RX channel used.
- ltc_edma_state_machine_t state_machine

  State machine.
- uint32_t state

  Internal state.
- const uint8_t * inData

  Input data.
- uint8_t * outData

  Output data.
- uint32_t size

  Size of input and output data in bytes.
- uint32_t modeReg

  LTC mode register.
- uint8_t * counter

  Input counter (updates on return)
- const uint8_t * key

  Input key to use for forward AES cipher.
- uint32_t keySize

  Size of the input key, in bytes.
- uint8_t * counterlast

  Output cipher of last counter, for chained CTR calls.
- uint32_t * szLeft

  Output number of bytes in left unused in counterlast block.
- uint32_t lastSize

  Last size.

21.9.2.1.0.1 Field Documentation

21.9.2.1.0.1.1 ltc_edma_callback_t ltc_edma_handle_t::callback

21.9.2.1.0.1.2 void∗ ltc_edma_handle_t::userData

21.9.2.1.0.1.3 edma_handle_t∗ ltc_edma_handle_t::inputFifoEdmaHandle

21.9.2.1.0.1.4 edma_handle_t∗ ltc_edma_handle_t::outputFifoEdmaHandle

21.9.2.1.0.1.5 ltc_edma_state_machine_t ltc_edma_handle_t::state_machine

21.9.2.1.0.1.6 uint32_t ltc_edma_handle_t::state

21.9.2.1.0.1.7 const uint8_t∗ ltc_edma_handle_t::inData

21.9.2.1.0.1.8 uint8_t∗ ltc_edma_handle_t::outData

21.9.2.1.0.1.9 uint32_t ltc_edma_handle_t::size

21.9.2.1.0.1.10 uint32_t ltc_edma_handle_t::modeReg

21.9.2.1.0.1.11 uint32_t ltc_edma_handle_t::keySize

Must be 16, 24, or 32.

21.9.2.1.0.1.12 uint8_t∗ ltc_edma_handle_t::counterlast

NULL can be passed if chained calls are not used.

21.9.2.1.0.1.13 uint32_t∗ ltc_edma_handle_t::szLeft

NULL can be passed if chained calls are not used.

21.9.2.1.0.1.14 uint32_t ltc_edma_handle_t::lastSize

21.9.3 Typedef Documentation

21.9.3.1 typedef void(∗ ltc_edma_callback_t)(LTC_Type ∗base, ltc_edma_handle_t ∗handle, status_t status, void ∗userData)

21.9.3.2 typedef status_t(∗ ltc_edma_state_machine_t)(LTC_Type ∗base, ltc_edma_handle_t ∗handle)

It is defined only for private usage inside LTC eDMA driver.

21.9.4 Function Documentation

21.9.4.1 void LTC_CreateHandleEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* ltc_edma_callback_t *callback,* void * *userData,* edma_handle_t * *inputFifoEdmaHandle,* edma_handle_t * *outputFifoEdmaHandle*)

Parameters

base	LTC module base address
handle	Pointer to ltc_edma_handle_t structure
callback	Callback function, NULL means no callback.
userData	Callback function parameter.
inputFifo-EdmaHandle	User requested eDMA handle for Input FIFO eDMA.
outputFifo-EdmaHandle	User requested eDMA handle for Output FIFO eDMA.

21.9.5 LTC eDMA DES driver

21.9.5.1 Overview

This section describes the programming interface of the LTC eDMA DES driver.

Functions

- status_t LTC_DES_EncryptEcbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using ECB block mode.
- status_t LTC_DES_DecryptEcbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using ECB block mode.
- status_t LTC_DES_EncryptCbcEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using CBC block mode.
- status_t LTC_DES_DecryptCbcEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using CBC block mode.
- status_t LTC_DES_EncryptCfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using CFB block mode.
- status_t LTC_DES_DecryptCfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using CFB block mode.
- status_t LTC_DES_EncryptOfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Encrypts DES using OFB block mode.
- status_t LTC_DES_DecryptOfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key[LTC_DES_KEY_SIZE])
  Decrypts DES using OFB block mode.
- status_t LTC_DES2_EncryptEcbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Encrypts triple DES using ECB block mode with two keys.
- status_t LTC_DES2_DecryptEcbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])
  Decrypts triple DES using ECB block mode with two keys.
- status_t LTC_DES2_EncryptCbcEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const

uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])

Encrypts triple DES using CBC block mode with two keys.

- status_t LTC_DES2_DecryptCbcEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])

  Decrypts triple DES using CBC block mode with two keys.
- status_t LTC_DES2_EncryptCfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])

  Encrypts triple DES using CFB block mode with two keys.
- status_t LTC_DES2_DecryptCfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])

  Decrypts triple DES using CFB block mode with two keys.
- status_t LTC_DES2_EncryptOfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])

  Encrypts triple DES using OFB block mode with two keys.
- status_t LTC_DES2_DecryptOfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE])

  Decrypts triple DES using OFB block mode with two keys.
- status_t LTC_DES3_EncryptEcbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Encrypts triple DES using ECB block mode with three keys.
- status_t LTC_DES3_DecryptEcbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Decrypts triple DES using ECB block mode with three keys.
- status_t LTC_DES3_EncryptCbcEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Encrypts triple DES using CBC block mode with three keys.
- status_t LTC_DES3_DecryptCbcEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Decrypts triple DES using CBC block mode with three keys.
- status_t LTC_DES3_EncryptCfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Encrypts triple DES using CFB block mode with three keys.
- status_t LTC_DES3_DecryptCfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const

uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

Decrypts triple DES using CFB block mode with three keys.

- status_t LTC_DES3_EncryptOfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *plaintext, uint8_t *ciphertext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Encrypts triple DES using OFB block mode with three keys.
- status_t LTC_DES3_DecryptOfbEDMA (LTC_Type *base, ltc_edma_handle_t *handle, const uint8_t *ciphertext, uint8_t *plaintext, uint32_t size, const uint8_t iv[LTC_DES_IV_SIZE], const uint8_t key1[LTC_DES_KEY_SIZE], const uint8_t key2[LTC_DES_KEY_SIZE], const uint8_t key3[LTC_DES_KEY_SIZE])

  Decrypts triple DES using OFB block mode with three keys.

21.9.5.2 Function Documentation

21.9.5.2.1 status_t LTC_DES_EncryptEcbEDMA (LTC_Type * *base*, ltc_edma_handle_t * *handle*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using ECB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
`out`	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.2 status_t LTC_DES_DecryptEcbEDMA (LTC_Type * *base*, ltc_edma_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using ECB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.3 status_t LTC_DES_EncryptCbcEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using CBC block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Ouput ciphertext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.4 status_t LTC_DES_DecryptCbcEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using CBC block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.5 status_t LTC_DES_EncryptCfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using CFB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
	size	Size of input data in bytes
	iv	Input initial block.
	key	Input key to use for encryption
out	*ciphertext*	Output ciphertext

Returns

Status from encrypt/decrypt operation

21.9.5.2.6 status_t LTC_DES_DecryptCfbEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using CFB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial block.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.7 status_t LTC_DES_EncryptOfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Encrypts DES using OFB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key	Input key to use for encryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.8 status_t LTC_DES_DecryptOfbEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key[LTC_DES_KEY_SIZE]*)

Decrypts DES using OFB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
`out`	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key	Input key to use for decryption

Returns

Status from encrypt/decrypt operation

21.9.5.2.9 status_t LTC_DES2_EncryptEcbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using ECB block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
`out`	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.10 status_t LTC_DES2_DecryptEcbEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using ECB block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.11 status_t LTC_DES2_EncryptCbcEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *plaintext,* uint8_t * *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CBC block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle

	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.12 status_t LTC_DES2_DecryptCbcEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CBC block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
`out`	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.13 status_t LTC_DES2_EncryptCfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.14 status_t LTC_DES2_DecryptCfbEDMA (LTC_Type * *base*, ltc_edma_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial block.
	key1	First input key for key bundle

	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.15 status_t LTC_DES2_EncryptOfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using OFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
`out`	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.16 status_t LTC_DES2_DecryptOfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using OFB block mode with two keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.17 status_t LTC_DES3_EncryptEcbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using ECB block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle

	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.18 status_t LTC_DES3_DecryptEcbEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using ECB block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
`out`	*plaintext*	Output plaintext
	size	Size of input and output data in bytes. Must be multiple of 8 bytes.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.19 status_t LTC_DES3_EncryptCbcEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *plaintext,* uint8_t * *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CBC block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.20 status_t LTC_DES3_DecryptCbcEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CBC block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input initial vector to combine with the first plaintext block. The iv does not need to be secret, but it must be unpredictable.

	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.21 status_t LTC_DES3_EncryptCfbEDMA (LTC_Type * *base*, ltc_edma_handle_t * *handle*, const uint8_t * *plaintext*, uint8_t * *ciphertext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using CFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
`out`	*ciphertext*	Output ciphertext
	size	Size of input and ouput data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.22 status_t LTC_DES3_DecryptCfbEDMA (LTC_Type * *base*, ltc_edma_handle_t * *handle*, const uint8_t * *ciphertext*, uint8_t * *plaintext*, uint32_t *size*, const uint8_t *iv[LTC_DES_IV_SIZE]*, const uint8_t *key1[LTC_DES_KEY_SIZE]*, const uint8_t *key2[LTC_DES_KEY_SIZE]*, const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using CFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
out	*plaintext*	Output plaintext
	size	Size of input data in bytes
	iv	Input initial block.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.23 status_t LTC_DES3_EncryptOfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *plaintext,* uint8_t ∗ *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Encrypts triple DES using OFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plaintext to encrypt
out	*ciphertext*	Output ciphertext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.

	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.5.2.24 status_t LTC_DES3_DecryptOfbEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_DES_IV_SIZE],* const uint8_t *key1[LTC_DES_KEY_SIZE],* const uint8_t *key2[LTC_DES_KEY_SIZE],* const uint8_t *key3[LTC_DES_KEY_SIZE]*)

Decrypts triple DES using OFB block mode with three keys.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input ciphertext to decrypt
`out`	*plaintext*	Output plaintext
	size	Size of input and output data in bytes
	iv	Input unique input vector. The OFB mode requires that the IV be unique for each execution of the mode under the given key.
	key1	First input key for key bundle
	key2	Second input key for key bundle
	key3	Third input key for key bundle

Returns

Status from encrypt/decrypt operation

21.9.6 LTC eDMA AES driver

21.9.6.1 Overview

This section describes the programming interface of the LTC eDMA AES driver.

Macros

- #define LTC_AES_DecryptCtrEDMA(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft) LTC_AES_CryptCtrEDMA(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft)
  AES CTR decrypt is mapped to the AES CTR generic operation.
- #define LTC_AES_EncryptCtrEDMA(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft) LTC_AES_CryptCtrEDMA(base, handle, input, output, size, counter, key, keySize, counterlast, szLeft)
  AES CTR encrypt is mapped to the AES CTR generic operation.

Functions

- status_t LTC_AES_EncryptEcbEDMA (LTC_Type ∗base, ltc_edma_handle_t ∗handle, const uint8_t ∗plaintext, uint8_t ∗ciphertext, uint32_t size, const uint8_t ∗key, uint32_t keySize)
  Encrypts AES using the ECB block mode.
- status_t LTC_AES_DecryptEcbEDMA (LTC_Type ∗base, ltc_edma_handle_t ∗handle, const uint8_t ∗ciphertext, uint8_t ∗plaintext, uint32_t size, const uint8_t ∗key, uint32_t keySize, ltc_aes_key_t keyType)
  Decrypts AES using ECB block mode.
- status_t LTC_AES_EncryptCbcEDMA (LTC_Type ∗base, ltc_edma_handle_t ∗handle, const uint8_t ∗plaintext, uint8_t ∗ciphertext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t ∗key, uint32_t keySize)
  Encrypts AES using CBC block mode.
- status_t LTC_AES_DecryptCbcEDMA (LTC_Type ∗base, ltc_edma_handle_t ∗handle, const uint8_t ∗ciphertext, uint8_t ∗plaintext, uint32_t size, const uint8_t iv[LTC_AES_IV_SIZE], const uint8_t ∗key, uint32_t keySize, ltc_aes_key_t keyType)
  Decrypts AES using CBC block mode.
- status_t LTC_AES_CryptCtrEDMA (LTC_Type ∗base, ltc_edma_handle_t ∗handle, const uint8_t ∗input, uint8_t ∗output, uint32_t size, uint8_t counter[LTC_AES_BLOCK_SIZE], const uint8_t ∗key, uint32_t keySize, uint8_t counterlast[LTC_AES_BLOCK_SIZE], uint32_t ∗szLeft)
  Encrypts or decrypts AES using CTR block mode.

21.9.6.2 Function Documentation

21.9.6.2.1 status_t LTC_AES_EncryptEcbEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *plaintext,* uint8_t * *ciphertext,* uint32_t *size,* const uint8_t * *key,* uint32_t *keySize*)

Encrypts AES using the ECB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plain text to encrypt
out	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	key	Input key to use for encryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

21.9.6.2.2 status_t LTC_AES_DecryptEcbEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *ciphertext,* uint8_t * *plaintext,* uint32_t *size,* const uint8_t * *key,* uint32_t *keySize,* ltc_aes_key_t *keyType*)

Decrypts AES using ECB block mode.

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input cipher text to decrypt
out	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	key	Input key.
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
	keyType	Input type of the key (allows to directly load decrypt key for AES ECB decrypt operation.)

Returns

Status from decrypt operation

21.9.6.2.3 status_t LTC_AES_EncryptCbcEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *plaintext,* uint8_t * *ciphertext,* uint32_t *size,* const uint8_t *iv[LTC_AES_IV_SIZE],* const uint8_t * *key,* uint32_t *keySize*)

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	plaintext	Input plain text to encrypt
`out`	*ciphertext*	Output cipher text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.
	key	Input key to use for encryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.

Returns

Status from encrypt operation

21.9.6.2.4 status_t LTC_AES_DecryptCbcEDMA (LTC_Type ∗ *base,* ltc_edma_handle_t ∗ *handle,* const uint8_t ∗ *ciphertext,* uint8_t ∗ *plaintext,* uint32_t *size,* const uint8_t *iv[LTC_AES_IV_SIZE],* const uint8_t ∗ *key,* uint32_t *keySize,* ltc_aes_key_t *keyType*)

Parameters

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	ciphertext	Input cipher text to decrypt
`out`	*plaintext*	Output plain text
	size	Size of input and output data in bytes. Must be multiple of 16 bytes.
	iv	Input initial vector to combine with the first input block.
	key	Input key to use for decryption
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
	keyType	Input type of the key (allows to directly load decrypt key for AES CBC decrypt operation.)

Returns

Status from decrypt operation

21.9.6.2.5 status_t LTC_AES_CryptCtrEDMA (LTC_Type * *base,* ltc_edma_handle_t * *handle,* const uint8_t * *input,* uint8_t * *output,* uint32_t *size,* uint8_t *counter[LTC_AES_BLOCK_SIZE],* const uint8_t * *key,* uint32_t *keySize,* uint8_t *counterlast[LTC_AES_BLOCK_SIZE],* uint32_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

	base	LTC peripheral base address
	handle	pointer to ltc_edma_handle_t structure which stores the transaction state.
	input	Input data for CTR block mode
`out`	*output*	Output data for CTR block mode
	size	Size of input and output data in bytes
`in,out`	*counter*	Input counter (updates on return)
	key	Input key to use for forward AES cipher
	keySize	Size of the input key, in bytes. Must be 16, 24, or 32.
`out`	*counterlast*	Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.
`out`	*szLeft*	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

Chapter 22
PIT: Periodic Interrupt Timer

22.1 Overview

The MCUXpresso SDK provides a driver for the Periodic Interrupt Timer (PIT) of MCUXpresso SDK devices.

22.2 Function groups

The PIT driver supports operating the module as a time counter.

22.2.1 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.

22.2.2 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.

22.2.3 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.

22.2.4 Status

Provides functions to get and clear the PIT status.

22.2.5 Interrupt

Provides functions to enable/disable PIT interrupts and get current enabled interrupts.

22.3 Typical use case

22.3.1 PIT tick example

Updates the PIT period and toggles an LED periodically.

```
int main(void)
{
    /* Structure of initialize PIT */
    pit_config_t pitConfig;

    /* Initialize and enable LED */
    LED_INIT();

    /* Board pin, clock, debug console init */
    BOARD_InitHardware();

    PIT_GetDefaultConfig(&pitConfig);

    /* Init pit module */
    PIT_Init(PIT, &pitConfig);

    /* Set timer period for channel 0 */
    PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, USEC_TO_COUNT(1000000U,
      PIT_SOURCE_CLOCK));

    /* Enable timer interrupts for channel 0 */
    PIT_EnableInterrupts(PIT, kPIT_Chnl_0,
      kPIT_TimerInterruptEnable);

    /* Enable at the NVIC */
    EnableIRQ(PIT_IRQ_ID);

    /* Start channel 0 */
    PRINTF("\r\nStarting channel No.0 ...");
    PIT_StartTimer(PIT, kPIT_Chnl_0);

    while (true)
    {
        /* Check whether occur interupt and toggle LED */
        if (true == pitIsrFlag)
        {
            PRINTF("\r\n Channel No.0 interrupt is occured !");
            LED_TOGGLE();
            pitIsrFlag = false;
        }
    }
}
```

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, 0))
  Version 2.0.0.

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.

22.4 Data Structure Documentation

22.4.1 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

22.5 Enumeration Type Documentation

22.5.1 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.

22.5.2 enum pit_interrupt_enable_t

Enumerator

kPIT_TimerInterruptEnable Timer interrupt enable.

22.5.3 enum pit_status_flags_t

Enumerator

kPIT_TimerFlag Timer flag.

22.6 Function Documentation

22.6.1 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

base	PIT peripheral base address
config	Pointer to the user's PIT config structure

22.6.2 void PIT_Deinit (PIT_Type ∗ *base*)

Parameters

base	PIT peripheral base address

22.6.3 static void PIT_GetDefaultConfig (pit_config_t ∗ *config*) [inline], [static]

The default values are as follows.

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

Parameters

config	Pointer to the onfiguration structure.

22.6.4 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

base	PIT peripheral base address
channel	Timer channel number which is chained with the previous timer
enable	Enable or disable chain. true: Current timer is chained with the previous timer. false: Timer doesn't chain with other timers.

22.6.5 static void PIT_EnableInterrupts (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number
mask	The interrupts to enable. This is a logical OR of members of the enumeration pit_-interrupt_enable_t

22.6.6 static void PIT_DisableInterrupts (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number
mask	The interrupts to disable. This is a logical OR of members of the enumeration pit_-interrupt_enable_t

22.6.7 static uint32_t PIT_GetEnabledInterrupts (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number

Returns

The enabled interrupts. This is the logical OR of members of the enumeration pit_interrupt_enable_t

22.6.8 static uint32_t PIT_GetStatusFlags (PIT_Type * *base*, pit_chnl_t *channel*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number

Returns

The status flags. This is the logical OR of members of the enumeration pit_status_flags_t

22.6.9 static void PIT_ClearStatusFlags (PIT_Type * *base*, pit_chnl_t *channel*, uint32_t *mask*) [inline], [static]

Parameters

base	PIT peripheral base address
channel	Timer channel number
mask	The status flags to clear. This is a logical OR of members of the enumeration pit_status_flags_t

22.6.10 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

base	PIT peripheral base address
channel	Timer channel number
count	Timer period in units of ticks

22.6.11 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

base	PIT peripheral base address
channel	Timer channel number

Returns

Current timer counting value in ticks

22.6.12 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

base	PIT peripheral base address
channel	Timer channel number.

22.6.13 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

base	PIT peripheral base address
channel	Timer channel number.

22.6.14 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

base	PIT peripheral base address

Returns

Current lifetime timer value

Chapter 23
PMC: Power Management Controller

23.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the Power Management Controller (PMC) module of MCUXpresso SDK devices. The PMC module contains internal voltage regulator, power on reset, low-voltage detect system, and high-voltage detect system.

Data Structures

- struct pmc_low_volt_detect_config_t
  Low-voltage Detect Configuration Structure. More...
- struct pmc_low_volt_warning_config_t
  Low-voltage Warning Configuration Structure. More...
- struct pmc_bandgap_buffer_config_t
  Bandgap Buffer configuration. More...

Enumerations

- enum pmc_low_volt_detect_volt_select_t {
  kPMC_LowVoltDetectLowTrip = 0U,
  kPMC_LowVoltDetectHighTrip = 1U }
  Low-voltage Detect Voltage Select.
- enum pmc_low_volt_warning_volt_select_t {
  kPMC_LowVoltWarningLowTrip = 0U,
  kPMC_LowVoltWarningMid1Trip = 1U,
  kPMC_LowVoltWarningMid2Trip = 2U,
  kPMC_LowVoltWarningHighTrip = 3U }
  Low-voltage Warning Voltage Select.
- enum pmc_vlp_freq_mode_t {
  kPMC_FreqRestrict = 0U,
  kPMC_FreqUnrestrict = 1U }
  VLPx Option.

Driver version

- #define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  PMC driver version.

Power Management Controller Control APIs

- void PMC_ConfigureLowVoltDetect (PMC_Type *base, const pmc_low_volt_detect_config_t *config)
  Configures the low-voltage detect setting.

- static bool PMC_GetLowVoltDetectFlag (PMC_Type ∗base)
  Gets the Low-voltage Detect Flag status.
- static void PMC_ClearLowVoltDetectFlag (PMC_Type ∗base)
  Acknowledges clearing the Low-voltage Detect flag.
- void PMC_ConfigureLowVoltWarning (PMC_Type ∗base, const pmc_low_volt_warning_config_t ∗config)
  Configures the low-voltage warning setting.
- static bool PMC_GetLowVoltWarningFlag (PMC_Type ∗base)
  Gets the Low-voltage Warning Flag status.
- static void PMC_ClearLowVoltWarningFlag (PMC_Type ∗base)
  Acknowledges the Low-voltage Warning flag.
- void PMC_ConfigureBandgapBuffer (PMC_Type ∗base, const pmc_bandgap_buffer_config_t ∗config)
  Configures the PMC bandgap.
- static bool PMC_GetPeriphIOIsolationFlag (PMC_Type ∗base)
  Gets the acknowledge Peripherals and I/O pads isolation flag.
- static void PMC_ClearPeriphIOIsolationFlag (PMC_Type ∗base)
  Acknowledges the isolation flag to Peripherals and I/O pads.
- static bool PMC_IsRegulatorInRunRegulation (PMC_Type ∗base)
  Gets the regulator regulation status.

23.2 Data Structure Documentation

23.2.1 struct pmc_low_volt_detect_config_t

Data Fields

- bool enableInt
  Enable interrupt when Low-voltage detect.
- bool enableReset
  Enable system reset when Low-voltage detect.
- pmc_low_volt_detect_volt_select_t voltSelect
  Low-voltage detect trip point voltage selection.

23.2.2 struct pmc_low_volt_warning_config_t

Data Fields

- bool enableInt
  Enable interrupt when low-voltage warning.
- pmc_low_volt_warning_volt_select_t voltSelect
  Low-voltage warning trip point voltage selection.

23.2.3 struct pmc_bandgap_buffer_config_t

Data Fields

- bool enable
  Enable bandgap buffer.

23.2.3.0.5.1 Field Documentation

23.2.3.0.5.1.1 bool pmc_bandgap_buffer_config_t::enable

23.3 Macro Definition Documentation

23.3.1 #define FSL_PMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))

Version 2.0.0.

23.4 Enumeration Type Documentation

23.4.1 enum pmc_low_volt_detect_volt_select_t

Enumerator

kPMC_LowVoltDetectLowTrip Low-trip point selected (VLVD = VLVDL)
kPMC_LowVoltDetectHighTrip High-trip point selected (VLVD = VLVDH)

23.4.2 enum pmc_low_volt_warning_volt_select_t

Enumerator

kPMC_LowVoltWarningLowTrip Low-trip point selected (VLVW = VLVW1)
kPMC_LowVoltWarningMid1Trip Mid 1 trip point selected (VLVW = VLVW2)
kPMC_LowVoltWarningMid2Trip Mid 2 trip point selected (VLVW = VLVW3)
kPMC_LowVoltWarningHighTrip High-trip point selected (VLVW = VLVW4)

23.4.3 enum pmc_vlp_freq_mode_t

Enumerator

kPMC_FreqRestrict Frequency is restricted in VLPx mode.
kPMC_FreqUnrestrict Frequency is unrestricted in VLPx mode.

23.5 Function Documentation

23.5.1 void PMC_ConfigureLowVoltDetect (PMC_Type * *base,* const pmc_low_volt_detect_config_t * *config*)

This function configures the low-voltage detect setting, including the trip point voltage setting, enables or disables the interrupt, enables or disables the system reset.

Parameters

base	PMC peripheral base address.
config	Low-voltage detect configuration structure.

23.5.2 static bool PMC_GetLowVoltDetectFlag (PMC_Type * *base*) [inline], [static]

This function reads the current LVDF status. If it returns 1, a low-voltage event is detected.

Parameters

base	PMC peripheral base address.

Returns

Current low-voltage detect flag
- true: Low-voltage detected
- false: Low-voltage not detected

23.5.3 static void PMC_ClearLowVoltDetectFlag (PMC_Type * *base*) [inline], [static]

This function acknowledges the low-voltage detection errors (write 1 to clear LVDF).

Parameters

base	PMC peripheral base address.

23.5.4 void PMC_ConfigureLowVoltWarning (PMC_Type * *base,* const pmc_low_volt_warning_config_t * *config*)

This function configures the low-voltage warning setting, including the trip point voltage setting and enabling or disabling the interrupt.

Parameters

base	PMC peripheral base address.
config	Low-voltage warning configuration structure.

23.5.5 static bool PMC_GetLowVoltWarningFlag (PMC_Type * *base*) [inline], [static]

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

Parameters

base	PMC peripheral base address.

Returns

Current LVWF status
- true: Low-voltage Warning Flag is set.
- false: the Low-voltage Warning does not happen.

23.5.6 static void PMC_ClearLowVoltWarningFlag (PMC_Type * *base*) [inline], [static]

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

base	PMC peripheral base address.

23.5.7 void PMC_ConfigureBandgapBuffer (PMC_Type * *base,* const pmc_bandgap_buffer_config_t * *config*)

This function configures the PMC bandgap, including the drive select and behavior in low-power mode.

Parameters

base	PMC peripheral base address.
config	Pointer to the configuration structure

23.5.8 static bool PMC_GetPeriphIOIsolationFlag (PMC_Type * *base*) [inline], [static]

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

Parameters

base	PMC peripheral base address.
base	Base address for current PMC instance.

Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

23.5.9 static void PMC_ClearPeriphIOIsolationFlag (PMC_Type * *base*) [inline], [static]

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.

Parameters

base	PMC peripheral base address.

23.5.10 static bool PMC_IsRegulatorInRunRegulation (PMC_Type * *base*) [inline], [static]

This function returns the regulator to run a regulation status. It provides the current status of the internal voltage regulator.

Parameters

base	PMC peripheral base address.
base	Base address for current PMC instance.

Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.

Chapter 24
PORT: Port Control and Interrupts

24.1 Overview

The MCUXpresso SDK provides a driver for the Port Control and Interrupts (PORT) module of MCUXpresso SDK devices.

24.2 Typical configuration use case

24.2.1 Input PORT configuration

```
/* Input pin PORT configuration */
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainDisable,
    kPORT_LowDriveStrength,
    kPORT_MuxAsGpio,
    kPORT_UnLockRegister,
};
/*  Sets the configuration */
PORT_SetPinConfig(PORTA, 4, &config);
```

24.2.2 I2C PORT Configuration

```
/*  I2C pin PORTconfiguration */
port_pin_config_t config = {
    kPORT_PullUp,
    kPORT_FastSlewRate,
    kPORT_PassiveFilterDisable,
    kPORT_OpenDrainEnable,
    kPORT_LowDriveStrength,
    kPORT_MuxAlt5,
    kPORT_UnLockRegister,
};
PORT_SetPinConfig(PORTE,24u,&config);
PORT_SetPinConfig(PORTE,25u,&config);
```

Data Structures

- struct port_pin_config_t
  PORT pin configuration structure. More...

Enumerations

- enum _port_pull {
  kPORT_PullDisable = 0U,
  kPORT_PullDown = 2U,
  kPORT_PullUp = 3U }

Internal resistor pull feature selection.

- enum _port_slew_rate {
  kPORT_FastSlewRate = 0U,
  kPORT_SlowSlewRate = 1U }
  Slew rate selection.
- enum _port_passive_filter_enable {
  kPORT_PassiveFilterDisable = 0U,
  kPORT_PassiveFilterEnable = 1U }
  Passive filter feature enable/disable.
- enum _port_drive_strength {
  kPORT_LowDriveStrength = 0U,
  kPORT_HighDriveStrength = 1U }
  Configures the drive strength.
- enum port_mux_t {
  kPORT_PinDisabledOrAnalog = 0U,
  kPORT_MuxAsGpio = 1U,
  kPORT_MuxAlt2 = 2U,
  kPORT_MuxAlt3 = 3U,
  kPORT_MuxAlt4 = 4U,
  kPORT_MuxAlt5 = 5U,
  kPORT_MuxAlt6 = 6U,
  kPORT_MuxAlt7 = 7U,
  kPORT_MuxAlt8 = 8U,
  kPORT_MuxAlt9 = 9U,
  kPORT_MuxAlt10 = 10U,
  kPORT_MuxAlt11 = 11U,
  kPORT_MuxAlt12 = 12U,
  kPORT_MuxAlt13 = 13U,
  kPORT_MuxAlt14 = 14U,
  kPORT_MuxAlt15 = 15U }
  Pin mux selection.
- enum port_interrupt_t {
  kPORT_InterruptOrDMADisabled = 0x0U,
  kPORT_DMARisingEdge = 0x1U,
  kPORT_DMAFallingEdge = 0x2U,
  kPORT_DMAEitherEdge = 0x3U,
  kPORT_InterruptLogicZero = 0x8U,
  kPORT_InterruptRisingEdge = 0x9U,
  kPORT_InterruptFallingEdge = 0xAU,
  kPORT_InterruptEitherEdge = 0xBU,
  kPORT_InterruptLogicOne = 0xCU }
  Configures the interrupt generation condition.

Driver version

- #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  Version 2.0.2.

Configuration

- static void PORT_SetPinConfig (PORT_Type *base, uint32_t pin, const port_pin_config_t *config)
  Sets the port PCR register.
- static void PORT_SetMultiplePinsConfig (PORT_Type *base, uint32_t mask, const port_pin_-config_t *config)
  Sets the port PCR register for multiple pins.
- static void PORT_SetPinMux (PORT_Type *base, uint32_t pin, port_mux_t mux)
  Configures the pin muxing.

Interrupt

- static void PORT_SetPinInterruptConfig (PORT_Type *base, uint32_t pin, port_interrupt_t config)
  Configures the port pin interrupt/DMA request.
- static void PORT_SetPinDriveStrength (PORT_Type *base, uint32_t pin, uint8_t strength)
  Configures the port pin drive strength.
- static uint32_t PORT_GetPinsInterruptFlags (PORT_Type *base)
  Reads the whole port status flag.
- static void PORT_ClearPinsInterruptFlags (PORT_Type *base, uint32_t mask)
  Clears the multiple pin interrupt status flag.

24.3 Data Structure Documentation

24.3.1 struct port_pin_config_t

Data Fields

- uint16_t pullSelect: 2
  No-pull/pull-down/pull-up select.
- uint16_t slewRate: 1
  Fast/slow slew rate Configure.
- uint16_t passiveFilterEnable: 1
  Passive filter enable/disable.
- uint16_t driveStrength: 1
  Fast/slow drive strength configure.
- uint16_t mux: 3
  Pin mux Configure.

24.4 Macro Definition Documentation

24.4.1 #define FSL_PORT_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))

24.5 Enumeration Type Documentation

24.5.1 enum _port_pull

Enumerator

kPORT_PullDisable Internal pull-up/down resistor is disabled.

kPORT_PullDown Internal pull-down resistor is enabled.
kPORT_PullUp Internal pull-up resistor is enabled.

24.5.2 enum _port_slew_rate

Enumerator

kPORT_FastSlewRate Fast slew rate is configured.
kPORT_SlowSlewRate Slow slew rate is configured.

24.5.3 enum _port_passive_filter_enable

Enumerator

kPORT_PassiveFilterDisable Passive input filter is disabled.
kPORT_PassiveFilterEnable Passive input filter is enabled.

24.5.4 enum _port_drive_strength

Enumerator

kPORT_LowDriveStrength Low-drive strength is configured.
kPORT_HighDriveStrength High-drive strength is configured.

24.5.5 enum port_mux_t

Enumerator

kPORT_PinDisabledOrAnalog Corresponding pin is disabled, but is used as an analog pin.
kPORT_MuxAsGpio Corresponding pin is configured as GPIO.
kPORT_MuxAlt2 Chip-specific.
kPORT_MuxAlt3 Chip-specific.
kPORT_MuxAlt4 Chip-specific.
kPORT_MuxAlt5 Chip-specific.
kPORT_MuxAlt6 Chip-specific.
kPORT_MuxAlt7 Chip-specific.
kPORT_MuxAlt8 Chip-specific.
kPORT_MuxAlt9 Chip-specific.
kPORT_MuxAlt10 Chip-specific.
kPORT_MuxAlt11 Chip-specific.

kPORT_MuxAlt12 Chip-specific.
kPORT_MuxAlt13 Chip-specific.
kPORT_MuxAlt14 Chip-specific.
kPORT_MuxAlt15 Chip-specific.

24.5.6 enum port_interrupt_t

Enumerator

kPORT_InterruptOrDMADisabled Interrupt/DMA request is disabled.
kPORT_DMARisingEdge DMA request on rising edge.
kPORT_DMAFallingEdge DMA request on falling edge.
kPORT_DMAEitherEdge DMA request on either edge.
kPORT_InterruptLogicZero Interrupt when logic zero.
kPORT_InterruptRisingEdge Interrupt on rising edge.
kPORT_InterruptFallingEdge Interrupt on falling edge.
kPORT_InterruptEitherEdge Interrupt on either edge.
kPORT_InterruptLogicOne Interrupt when logic one.

24.6 Function Documentation

24.6.1 static void PORT_SetPinConfig (PORT_Type * *base*, uint32_t *pin*, const port_pin_config_t * *config*) [inline], [static]

This is an example to define an input pin or output pin PCR configuration.

```
* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*      kPORT_PullUp,
*      kPORT_FastSlewRate,
*      kPORT_PassiveFilterDisable,
*      kPORT_OpenDrainDisable,
*      kPORT_LowDriveStrength,
*      kPORT_MuxAsGpio,
*      kPORT_UnLockRegister,
* };
*
```

Parameters

base	PORT peripheral base pointer.

pin	PORT pin number.
config	PORT PCR register configuration structure.

24.6.2 static void PORT_SetMultiplePinsConfig (PORT_Type * *base*, uint32_t *mask*, const port_pin_config_t * *config*) [inline], [static]

This is an example to define input pins or output pins PCR configuration.

```
* // Define a digital input pin PCR configuration
* port_pin_config_t config = {
*      kPORT_PullUp ,
*      kPORT_PullEnable,
*      kPORT_FastSlewRate,
*      kPORT_PassiveFilterDisable,
*      kPORT_OpenDrainDisable,
*      kPORT_LowDriveStrength,
*      kPORT_MuxAsGpio,
*      kPORT_UnlockRegister,
* };
*
```

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.
config	PORT PCR register configuration structure.

24.6.3 static void PORT_SetPinMux (PORT_Type * *base*, uint32_t *pin*, port_mux_t *mux*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
mux	pin muxing slot selection. • kPORT_PinDisabledOrAnalog: Pin disabled or work in analog function. • kPORT_MuxAsGpio : Set as GPIO. • kPORT_MuxAlt2 : chip-specific. • kPORT_MuxAlt3 : chip-specific. • kPORT_MuxAlt4 : chip-specific. • kPORT_MuxAlt5 : chip-specific. • kPORT_MuxAlt6 : chip-specific. • kPORT_MuxAlt7 : chip-specific. : This function is NOT recommended to use together with the PORT_SetPinsConfig, because the PORT_SetPinsConfig need to configure the pin mux anyway (Otherwise the pin mux is reset to zero : kPORT_PinDisabledOrAnalog). This function is recommended to use to reset the pin mux

24.6.4 static void PORT_SetPinInterruptConfig (PORT_Type * *base*, uint32_t *pin*, port_interrupt_t *config*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.
config	PORT pin interrupt configuration. • kPORT_InterruptOrDMADisabled: Interrupt/DMA request disabled. • kPORT_DMARisingEdge : DMA request on rising edge(if the DMA requests exit). • kPORT_DMAFallingEdge: DMA request on falling edge(if the DMA requests exit). • kPORT_DMAEitherEdge : DMA request on either edge(if the DMA requests exit). • #kPORT_FlagRisingEdge : Flag sets on rising edge(if the Flag states exit). • #kPORT_FlagFallingEdge : Flag sets on falling edge(if the Flag states exit). • #kPORT_FlagEitherEdge : Flag sets on either edge(if the Flag states exit). • kPORT_InterruptLogicZero : Interrupt when logic zero. • kPORT_InterruptRisingEdge : Interrupt on rising edge. • kPORT_InterruptFallingEdge: Interrupt on falling edge. • kPORT_InterruptEitherEdge : Interrupt on either edge. • kPORT_InterruptLogicOne : Interrupt when logic one. • #kPORT_ActiveHighTriggerOutputEnable : Enable active high-trigger output (if the trigger states exit). • #kPORT_ActiveLowTriggerOutputEnable : Enable active low-trigger output (if the trigger states exit).

24.6.5 static void PORT_SetPinDriveStrength (PORT_Type * *base*, uint32_t *pin*, uint8_t *strength*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
pin	PORT pin number.

config	PORT pin drive strength • kPORT_LowDriveStrength = 0U - Low-drive strength is configured. • kPORT_HighDriveStrength = 1U - High-drive strength is configured.

24.6.6 static uint32_t PORT_GetPinsInterruptFlags (PORT_Type * *base*) [inline], [static]

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

base	PORT peripheral base pointer.

Returns

Current port interrupt status flags, for example, 0x00010001 means the pin 0 and 16 have the interrupt.

24.6.7 static void PORT_ClearPinsInterruptFlags (PORT_Type * *base,* uint32_t *mask*) [inline], [static]

Parameters

base	PORT peripheral base pointer.
mask	PORT pin number macro.

Chapter 25
RCM: Reset Control Module Driver

25.1 Overview

The MCUXpresso SDK provides a Peripheral driver for the Reset Control Module (RCM) module of MCUXpresso SDK devices.

Data Structures

- struct rcm_reset_pin_filter_config_t
  Reset pin filter configuration. More...

Enumerations

- enum rcm_reset_source_t {
  kRCM_SourceWakeup = RCM_SRS0_WAKEUP_MASK,
  kRCM_SourceLvd = RCM_SRS0_LVD_MASK,
  kRCM_SourceLoc = RCM_SRS0_LOC_MASK,
  kRCM_SourceWdog = RCM_SRS0_WDOG_MASK,
  kRCM_SourcePin = RCM_SRS0_PIN_MASK,
  kRCM_SourcePor = RCM_SRS0_POR_MASK,
  kRCM_SourceLockup = RCM_SRS1_LOCKUP_MASK << 8U,
  kRCM_SourceSw = RCM_SRS1_SW_MASK << 8U,
  kRCM_SourceMdmap = RCM_SRS1_MDM_AP_MASK << 8U,
  kRCM_SourceSackerr = RCM_SRS1_SACKERR_MASK << 8U }
  System Reset Source Name definitions.
- enum rcm_run_wait_filter_mode_t {
  kRCM_FilterDisable = 0U,
  kRCM_FilterBusClock = 1U,
  kRCM_FilterLpoClock = 2U }
  Reset pin filter select in Run and Wait modes.

Driver version

- #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))
  RCM driver version 2.0.1.

Reset Control Module APIs

- static uint32_t RCM_GetPreviousResetSources (RCM_Type *base)
  Gets the reset source status which caused a previous reset.
- void RCM_ConfigureResetPinFilter (RCM_Type *base, const rcm_reset_pin_filter_config_t *config)
  Configures the reset pin filter.

25.2 Data Structure Documentation

25.2.1 struct rcm_reset_pin_filter_config_t

Data Fields

- bool enableFilterInStop
  Reset pin filter select in stop mode.
- rcm_run_wait_filter_mode_t filterInRunWait
  Reset pin filter in run/wait mode.
- uint8_t busClockFilterCount
  Reset pin bus clock filter width.

25.2.1.0.5.2 Field Documentation

25.2.1.0.5.2.1 bool rcm_reset_pin_filter_config_t::enableFilterInStop

25.2.1.0.5.2.2 rcm_run_wait_filter_mode_t rcm_reset_pin_filter_config_t::filterInRunWait

25.2.1.0.5.2.3 uint8_t rcm_reset_pin_filter_config_t::busClockFilterCount

25.3 Macro Definition Documentation

25.3.1 #define FSL_RCM_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

25.4 Enumeration Type Documentation

25.4.1 enum rcm_reset_source_t

Enumerator

kRCM_SourceWakeup Low-leakage wakeup reset.
kRCM_SourceLvd Low-voltage detect reset.
kRCM_SourceLoc Loss of clock reset.
kRCM_SourceWdog Watchdog reset.
kRCM_SourcePin External pin reset.
kRCM_SourcePor Power on reset.
kRCM_SourceLockup Core lock up reset.
kRCM_SourceSw Software reset.
kRCM_SourceMdmap MDM-AP system reset.
kRCM_SourceSackerr Parameter could get all reset flags.

25.4.2 enum rcm_run_wait_filter_mode_t

Enumerator

kRCM_FilterDisable All filtering disabled.

kRCM_FilterBusClock Bus clock filter enabled.
kRCM_FilterLpoClock LPO clock filter enabled.

25.5 Function Documentation

25.5.1 static uint32_t RCM_GetPreviousResetSources (RCM_Type * *base*) [inline], [static]

This function gets the current reset source status. Use source masks defined in the rcm_reset_source_t to get the desired source status.

This is an example.

```
uint32_t resetStatus;

// To get all reset source statuses.
resetStatus = RCM_GetPreviousResetSources(RCM) & kRCM_SourceAll;

// To test whether the MCU is reset using Watchdog.
resetStatus = RCM_GetPreviousResetSources(RCM) &
      kRCM_SourceWdog;

// To test multiple reset sources.
resetStatus = RCM_GetPreviousResetSources(RCM) & (
      kRCM_SourceWdog | kRCM_SourcePin);
```

Parameters

base	RCM peripheral base address.

Returns

All reset source status bit map.

25.5.2 void RCM_ConfigureResetPinFilter (RCM_Type * *base,* const rcm_reset_pin_filter_config_t * *config*)

This function sets the reset pin filter including the filter source, filter width, and so on.

Parameters

base	RCM peripheral base address.

config	Pointer to the configuration structure.

Chapter 26
RTC: Real Time Clock

26.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC) of MCUXpresso SDK devices.

26.2 Function groups

The RTC driver supports operating the module as a time counter.

26.2.1 Initialization and deinitialization

The function RTC_Init() initializes the RTC with specified configurations. The function RTC_GetDefaultConfig() gets the default configurations.

The function RTC_Deinit() disables the RTC timer and disables the module clock.

26.2.2 Set & Get Datetime

The function RTC_SetDatetime() sets the timer period in seconds. Users pass in the details in date & time format by using the below data structure.

```
typedef struct _rtc_datetime
{
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
} rtc_datetime_t;
```

The function RTC_GetDatetime() reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

26.2.3 Set & Get Alarm

The function RTC_SetAlarm() sets the alarm time period in seconds. Users pass in the details in date & time format by using the datetime data structure.

The function RTC_GetAlarm() reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

26.2.4 Start & Stop timer

The function RTC_StartTimer() starts the RTC time counter.

The function RTC_StopTimer() stops the RTC time counter.

26.2.5 Status

Provides functions to get and clear the RTC status.

26.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

26.2.7 RTC Oscillator

Some SoC's allow control of the RTC oscillator through the RTC module.

The function RTC_SetOscCapLoad() allows the user to modify the capacitor load configuration of the RTC oscillator.

26.2.8 Monotonic Counter

Some SoC's have a 64-bit Monotonic counter available in the RTC module.

The function RTC_SetMonotonicCounter() writes a 64-bit to the counter.

The function RTC_GetMonotonicCounter() reads the monotonic counter and returns the 64-bit counter value to the user.

The function RTC_IncrementMonotonicCounter() increments the Monotonic Counter by one.

26.3 Typical use case

26.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm.

```
int main(void)
{
    uint32_t sec;
    uint32_t currSeconds;
    rtc_datetime_t date;
    rtc_config_t rtcConfig;

    /* Board pin, clock, debug console init */
```

```
BOARD_InitHardware();
/* Init RTC */
RTC_GetDefaultConfig(&rtcConfig);
RTC_Init(RTC, &rtcConfig);
/* Select RTC clock source */
BOARD_SetRtcClockSource();

PRINTF("RTC example: set up time to wake up an alarm\r\n");

/* Set a start date time and start RT */
date.year = 2014U;
date.month = 12U;
date.day = 25U;
date.hour = 19U;
date.minute = 0;
date.second = 0;

/* RTC time counter has to be stopped before setting the date & time in the TSR register */
RTC_StopTimer(RTC);

/* Set RTC time to default */
RTC_SetDatetime(RTC, &date);

/* Enable RTC alarm interrupt */
RTC_EnableInterrupts(RTC, kRTC_AlarmInterruptEnable);

/* Enable at the NVIC */
EnableIRQ(RTC_IRQn);

/* Start the RTC time counter */
RTC_StartTimer(RTC);

/* This loop will set the RTC alarm */
while (1)
{
    busyWait = true;
    /* Get date time */
    RTC_GetDatetime(RTC, &date);

    /* print default time */
    PRINTF("Current datetime: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n", date.
  year, date.month, date.day, date.hour,
            date.minute, date.second);

    /* Get alarm time from the user */
    sec = 0;
    PRINTF("Input the number of second to wait for alarm \r\n");
    PRINTF("The second must be positive value\r\n");
    while (sec < 1)
    {
        SCANF("%d", &sec);
    }

    /* Read the RTC seconds register to get current time in seconds */
    currSeconds = RTC->TSR;

    /* Add alarm seconds to current time */
    currSeconds += sec;

    /* Set alarm time in seconds */
    RTC->TAR = currSeconds;

    /* Get alarm time */
    RTC_GetAlarm(RTC, &date);

    /* Print alarm time */
    PRINTF("Alarm will occur at: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n", date.
  year, date.month, date.day,
```

```
                date.hour, date.minute, date.second);

        /* Wait until alarm occurs */
        while (busyWait)
        {
        }

        PRINTF("\r\n Alarm occurs !!!! ");
    }
}
```

Data Structures

- struct rtc_datetime_t
  Structure is used to hold the date and time. More...
- struct rtc_config_t
  RTC config structure. More...

Enumerations

- enum rtc_interrupt_enable_t {
  kRTC_TimeInvalidInterruptEnable = (1U << 0U),
  kRTC_TimeOverflowInterruptEnable = (1U << 1U),
  kRTC_AlarmInterruptEnable = (1U << 2U),
  kRTC_SecondsInterruptEnable = (1U << 3U) }
  List of RTC interrupts.
- enum rtc_status_flags_t {
  kRTC_TimeInvalidFlag = (1U << 0U),
  kRTC_TimeOverflowFlag = (1U << 1U),
  kRTC_AlarmFlag = (1U << 2U) }
  List of RTC flags.
- enum rtc_osc_cap_load_t {
  kRTC_Capacitor_2p = RTC_CR_SC2P_MASK,
  kRTC_Capacitor_4p = RTC_CR_SC4P_MASK,
  kRTC_Capacitor_8p = RTC_CR_SC8P_MASK,
  kRTC_Capacitor_16p = RTC_CR_SC16P_MASK }
  List of RTC Oscillator capacitor load settings.

Functions

- static void RTC_SetClockSource (RTC_Type *base)
  Set RTC clock source.
- static void RTC_SetOscCapLoad (RTC_Type *base, uint32_t capLoad)
  This function sets the specified capacitor configuration for the RTC oscillator.
- static void RTC_Reset (RTC_Type *base)
  Performs a software reset on the RTC module.

Driver version

- #define FSL_RTC_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  Version 2.0.0.

Initialization and deinitialization

- void RTC_Init (RTC_Type *base, const rtc_config_t *config)
  Ungates the RTC clock and configures the peripheral for basic operation.
- static void RTC_Deinit (RTC_Type *base)
  Stops the timer and gate the RTC clock.
- void RTC_GetDefaultConfig (rtc_config_t *config)
  Fills in the RTC config struct with the default settings.

Current Time & Alarm

- status_t RTC_SetDatetime (RTC_Type *base, const rtc_datetime_t *datetime)
  Sets the RTC date and time according to the given time structure.
- void RTC_GetDatetime (RTC_Type *base, rtc_datetime_t *datetime)
  Gets the RTC time and stores it in the given time structure.
- status_t RTC_SetAlarm (RTC_Type *base, const rtc_datetime_t *alarmTime)
  Sets the RTC alarm time.
- void RTC_GetAlarm (RTC_Type *base, rtc_datetime_t *datetime)
  Returns the RTC alarm time.

Interrupt Interface

- void RTC_EnableInterrupts (RTC_Type *base, uint32_t mask)
  Enables the selected RTC interrupts.
- void RTC_DisableInterrupts (RTC_Type *base, uint32_t mask)
  Disables the selected RTC interrupts.
- uint32_t RTC_GetEnabledInterrupts (RTC_Type *base)
  Gets the enabled RTC interrupts.

Status Interface

- uint32_t RTC_GetStatusFlags (RTC_Type *base)
  Gets the RTC status flags.
- void RTC_ClearStatusFlags (RTC_Type *base, uint32_t mask)
  Clears the RTC status flags.

Timer Start and Stop

- static void RTC_StartTimer (RTC_Type *base)
  Starts the RTC time counter.
- static void RTC_StopTimer (RTC_Type *base)
  Stops the RTC time counter.

26.4 Data Structure Documentation

26.4.1 struct 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.

26.4.1.0.5.3 Field Documentation

26.4.1.0.5.3.1 uint16_t rtc_datetime_t::year

26.4.1.0.5.3.2 uint8_t rtc_datetime_t::month

26.4.1.0.5.3.3 uint8_t rtc_datetime_t::day

26.4.1.0.5.3.4 uint8_t rtc_datetime_t::hour

26.4.1.0.5.3.5 uint8_t rtc_datetime_t::minute

26.4.1.0.5.3.6 uint8_t rtc_datetime_t::second

26.4.2 struct rtc_config_t

This structure holds the configuration settings for the RTC peripheral. To initialize this structure to reasonable defaults, call the RTC_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 wakeupSelect
  true: Wakeup pin outputs the 32 KHz clock; false:Wakeup pin used to wakeup the chip
- bool updateMode
  true: Registers can be written even when locked under certain conditions, false: No writes allowed when registers are locked
- bool supervisorAccess
  true: Non-supervisor accesses are allowed; false: Non-supervisor accesses are not supported
- uint32_t compensationInterval
  Compensation interval that is written to the CIR field in RTC TCR Register.
- uint32_t compensationTime
  Compensation time that is written to the TCR field in RTC TCR Register.

26.5 Enumeration Type Documentation

26.5.1 enum rtc_interrupt_enable_t

Enumerator

kRTC_TimeInvalidInterruptEnable Time invalid interrupt.
kRTC_TimeOverflowInterruptEnable Time overflow interrupt.
kRTC_AlarmInterruptEnable Alarm interrupt.
kRTC_SecondsInterruptEnable Seconds interrupt.

26.5.2 enum rtc_status_flags_t

Enumerator

kRTC_TimeInvalidFlag Time invalid flag.
kRTC_TimeOverflowFlag Time overflow flag.
kRTC_AlarmFlag Alarm flag.

26.5.3 enum rtc_osc_cap_load_t

Enumerator

kRTC_Capacitor_2p 2 pF capacitor load
kRTC_Capacitor_4p 4 pF capacitor load
kRTC_Capacitor_8p 8 pF capacitor load
kRTC_Capacitor_16p 16 pF capacitor load

26.6 Function Documentation

26.6.1 void RTC_Init (RTC_Type ∗ *base,* const rtc_config_t ∗ *config*)

This function issues a software reset if the timer invalid flag is set.

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

base	RTC peripheral base address
config	Pointer to the user's RTC configuration structure.

26.6.2 static void RTC_Deinit (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

26.6.3 void RTC_GetDefaultConfig (rtc_config_t * *config*)

The default values are as follows.

```
*    config->wakeupSelect = false;
*    config->updateMode = false;
*    config->supervisorAccess = false;
*    config->compensationInterval = 0;
*    config->compensationTime = 0;
*
```

Parameters

config	Pointer to the user's RTC configuration structure.

26.6.4 status_t RTC_SetDatetime (RTC_Type * *base,* const rtc_datetime_t * *datetime*)

The RTC counter must be stopped prior to calling this function because writes to the RTC seconds register fail if the RTC counter is running.

Parameters

base	RTC peripheral base address

datetime	Pointer to the structure where the date and time details are stored.

Returns

kStatus_Success: Success in setting the time and starting the RTC kStatus_InvalidArgument: Error because the datetime format is incorrect

26.6.5 void RTC_GetDatetime (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to the structure where the date and time details are stored.

26.6.6 status_t RTC_SetAlarm (RTC_Type * *base,* const rtc_datetime_t * *alarmTime*)

The function 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

base	RTC peripheral base address
alarmTime	Pointer to the structure where the alarm time is stored.

Returns

kStatus_Success: success in setting the RTC alarm kStatus_InvalidArgument: Error because the alarm datetime format is incorrect kStatus_Fail: Error because the alarm time has already passed

26.6.7 void RTC_GetAlarm (RTC_Type * *base,* rtc_datetime_t * *datetime*)

Parameters

base	RTC peripheral base address
datetime	Pointer to the structure where the alarm date and time details are stored.

26.6.8 void RTC_EnableInterrupts (RTC_Type ∗ *base,* uint32_t *mask*)

Parameters

base	RTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

26.6.9 void RTC_DisableInterrupts (RTC_Type ∗ *base,* uint32_t *mask*)

Parameters

base	RTC peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration rtc_interrupt_enable_t

26.6.10 uint32_t RTC_GetEnabledInterrupts (RTC_Type ∗ *base*)

Parameters

base	RTC peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration rtc_interrupt_enable_t

26.6.11 uint32_t RTC_GetStatusFlags (RTC_Type ∗ *base*)

Parameters

base	RTC peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration rtc_status_flags_t

26.6.12 void RTC_ClearStatusFlags (RTC_Type * *base,* uint32_t *mask*)

Parameters

base	RTC peripheral base address
mask	The status flags to clear. This is a logical OR of members of the enumeration rtc_-status_flags_t

26.6.13 static void RTC_SetClockSource (RTC_Type * *base*) [inline], [static]

Parameters

base	RTC peripheral base address

Note

After setting this bit, wait the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

26.6.14 static void RTC_StartTimer (RTC_Type * *base*) [inline], [static]

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

Parameters

base	RTC peripheral base address

26.6.15 static void RTC_StopTimer (RTC_Type * *base*) [inline], [static]

RTC's seconds register can be written to only when the timer is stopped.

Parameters

base	RTC peripheral base address

26.6.16 static void RTC_SetOscCapLoad (RTC_Type * *base,* uint32_t *capLoad*) [inline], [static]

Parameters

base	RTC peripheral base address
capLoad	Oscillator loads to enable. This is a logical OR of members of the enumeration rtc_osc_cap_load_t

26.6.17 static void RTC_Reset (RTC_Type * *base*) [inline], [static]

This resets all RTC registers except for the SWR bit and the RTC_WAR and RTC_RAR registers. The SWR bit is cleared by software explicitly clearing it.

Parameters

base	RTC peripheral base address

Chapter 27
SIM: System Integration Module Driver

27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Integration Module (SIM) of MCUXpresso SDK devices.

Data Structures

- struct sim_uid_t
  Unique ID. More...

Enumerations

- enum _sim_flash_mode {
  kSIM_FlashDisableInWait = SIM_FCFG1_FLASHDOZE_MASK,
  kSIM_FlashDisable = SIM_FCFG1_FLASHDIS_MASK }
  Flash enable mode.

Functions

- void SIM_GetUniqueId (sim_uid_t *uid)
  Gets the unique identification register value.
- static void SIM_SetFlashMode (uint8_t mode)
  Sets the flash enable mode.

Driver version

- #define FSL_SIM_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))
  Driver version 2.0.0.

27.2 Data Structure Documentation

27.2.1 struct sim_uid_t

Data Fields

- uint32_t MH
  UIDMH.
- uint32_t ML
  UIDML.
- uint32_t L
  UIDL.

27.2.1.0.5.4 Field Documentation

27.2.1.0.5.4.1 uint32_t sim_uid_t::MH

27.2.1.0.5.4.2 uint32_t sim_uid_t::ML

27.2.1.0.5.4.3 uint32_t sim_uid_t::L

27.3 Enumeration Type Documentation

27.3.1 enum _sim_flash_mode

Enumerator

kSIM_FlashDisableInWait Disable flash in wait mode.
kSIM_FlashDisable Disable flash in normal mode.

27.4 Function Documentation

27.4.1 void SIM_GetUniqueId (sim_uid_t ∗ *uid*)

Parameters

uid	Pointer to the structure to save the UID value.

27.4.2 static void SIM_SetFlashMode (uint8_t *mode*) [inline], [static]

Parameters

mode	The mode to set; see _sim_flash_mode for mode details.

Chapter 28
SMC: System Mode Controller Driver

28.1 Overview

The MCUXpresso SDK provides a peripheral driver for the System Mode Controller (SMC) module of MCUXpresso SDK devices. The SMC module sequences the system in and out of all low-power stop and run modes.

API functions are provided to configure the system for working in a dedicated power mode. For different power modes, SMC_SetPowerModexxx() function accepts different parameters. System power mode state transitions are not available between power modes. For details about available transitions, see the power mode transitions section in the SoC reference manual.

28.2 Typical use case

28.2.1 Enter wait or stop modes

SMC driver provides APIs to set MCU to different wait modes and stop modes. Pre and post functions are used for setting the modes. The pre functions and post functions are used as follows.

1. Disable/enable the interrupt through PRIMASK. This is an example use case. The application sets the wakeup interrupt and calls SMC function SMC_SetPowerModeStop to set the MCU to STOP mode, but the wakeup interrupt happens so quickly that the ISR completes before the function SMC_SetPowerModeStop. As a result, the MCU enters the STOP mode and never is woken up by the interrupt. In this use case, the application first disables the interrupt through PRIMASK, sets the wakeup interrupt, and enters the STOP mode. After wakeup, enable the interrupt through PRIMASK. The MCU can still be woken up by disabling the interrupt through PRIMASK. The pre and post functions handle the PRIMASK.
2. Disable/enable the flash speculation. When entering stop modes, the flash speculation might be interrupted. As a result, pre functions disable the flash speculation and post functions enable it.

```
SMC_PreEnterStopModes();

/* Enable the wakeup interrupt here. */

SMC_SetPowerModeStop(SMC, kSMC_PartialStop);

SMC_PostExitStopModes();
```

Data Structures

- struct smc_power_mode_lls_config_t
  SMC Low-Leakage Stop power mode configuration. More...
- struct smc_power_mode_vlls_config_t
  SMC Very Low-Leakage Stop power mode configuration. More...

Enumerations

- enum smc_power_mode_protection_t {
  kSMC_AllowPowerModeVlls = SMC_PMPROT_AVLLS_MASK,
  kSMC_AllowPowerModeLls = SMC_PMPROT_ALLS_MASK,
  kSMC_AllowPowerModeVlp = SMC_PMPROT_AVLP_MASK,
  kSMC_AllowPowerModeAll }
  Power Modes Protection.
- enum smc_power_state_t {
  kSMC_PowerStateRun = 0x01U << 0U,
  kSMC_PowerStateStop = 0x01U << 1U,
  kSMC_PowerStateVlpr = 0x01U << 2U,
  kSMC_PowerStateVlpw = 0x01U << 3U,
  kSMC_PowerStateVlps = 0x01U << 4U,
  kSMC_PowerStateLls = 0x01U << 5U,
  kSMC_PowerStateVlls = 0x01U << 6U }
  Power Modes in PMSTAT.
- enum smc_run_mode_t {
  kSMC_RunNormal = 0U,
  kSMC_RunVlpr = 2U }
  Run mode definition.
- enum smc_stop_mode_t {
  kSMC_StopNormal = 0U,
  kSMC_StopVlps = 2U,
  kSMC_StopLls = 3U,
  kSMC_StopVlls = 4U }
  Stop mode definition.
- enum smc_stop_submode_t {
  kSMC_StopSub0 = 0U,
  kSMC_StopSub1 = 1U,
  kSMC_StopSub2 = 2U,
  kSMC_StopSub3 = 3U }
  VLLS/LLS stop sub mode definition.
- enum smc_partial_stop_option_t {
  kSMC_PartialStop = 0U,
  kSMC_PartialStop1 = 1U,
  kSMC_PartialStop2 = 2U }
  Partial STOP option.
- enum _smc_status { kStatus_SMC_StopAbort = MAKE_STATUS(kStatusGroup_POWER, 0) }
  SMC configuration status.

Driver version

- #define FSL_SMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))
  SMC driver version 2.0.3.

System mode controller APIs

- static void SMC_SetPowerModeProtection (SMC_Type *base, uint8_t allowedModes)
  Configures all power mode protection settings.
- static smc_power_state_t SMC_GetPowerModeState (SMC_Type *base)
  Gets the current power mode status.
- void SMC_PreEnterStopModes (void)
  Prepares to enter stop modes.
- void SMC_PostExitStopModes (void)
  Recovers after wake up from stop modes.
- static void SMC_PreEnterWaitModes (void)
  Prepares to enter wait modes.
- static void SMC_PostExitWaitModes (void)
  Recovers after wake up from stop modes.
- status_t SMC_SetPowerModeRun (SMC_Type *base)
  Configures the system to RUN power mode.
- status_t SMC_SetPowerModeWait (SMC_Type *base)
  Configures the system to WAIT power mode.
- status_t SMC_SetPowerModeStop (SMC_Type *base, smc_partial_stop_option_t option)
  Configures the system to Stop power mode.
- status_t SMC_SetPowerModeVlpr (SMC_Type *base)
  Configures the system to VLPR power mode.
- status_t SMC_SetPowerModeVlpw (SMC_Type *base)
  Configures the system to VLPW power mode.
- status_t SMC_SetPowerModeVlps (SMC_Type *base)
  Configures the system to VLPS power mode.
- status_t SMC_SetPowerModeLls (SMC_Type *base, const smc_power_mode_lls_config_t *config)
  Configures the system to LLS power mode.
- status_t SMC_SetPowerModeVlls (SMC_Type *base, const smc_power_mode_vlls_config_t *config)
  Configures the system to VLLS power mode.

28.3 Data Structure Documentation

28.3.1 struct smc_power_mode_lls_config_t

Data Fields

- smc_stop_submode_t subMode
  Low-leakage Stop sub-mode.

28.3.2 struct smc_power_mode_vlls_config_t

Data Fields

- smc_stop_submode_t subMode
  Very Low-leakage Stop sub-mode.
- bool enablePorDetectInVlls0

Enable Power on reset detect in VLLS mode.

- bool enableRam2InVlls2

  Enable RAM2 power in VLLS2.

28.4 Macro Definition Documentation

28.4.1 #define FSL_SMC_DRIVER_VERSION (MAKE_VERSION(2, 0, 3))

28.5 Enumeration Type Documentation

28.5.1 enum smc_power_mode_protection_t

Enumerator

kSMC_AllowPowerModeVlls Allow Very-low-leakage Stop Mode.
kSMC_AllowPowerModeLls Allow Low-leakage Stop Mode.
kSMC_AllowPowerModeVlp Allow Very-Low-power Mode.
kSMC_AllowPowerModeAll Allow all power mode.

28.5.2 enum smc_power_state_t

Enumerator

kSMC_PowerStateRun 0000_0001 - Current power mode is RUN
kSMC_PowerStateStop 0000_0010 - Current power mode is STOP
kSMC_PowerStateVlpr 0000_0100 - Current power mode is VLPR
kSMC_PowerStateVlpw 0000_1000 - Current power mode is VLPW
kSMC_PowerStateVlps 0001_0000 - Current power mode is VLPS
kSMC_PowerStateLls 0010_0000 - Current power mode is LLS
kSMC_PowerStateVlls 0100_0000 - Current power mode is VLLS

28.5.3 enum smc_run_mode_t

Enumerator

kSMC_RunNormal Normal RUN mode.
kSMC_RunVlpr Very-low-power RUN mode.

28.5.4 enum smc_stop_mode_t

Enumerator

kSMC_StopNormal Normal STOP mode.

kSMC_StopVlps Very-low-power STOP mode.
kSMC_StopLls Low-leakage Stop mode.
kSMC_StopVlls Very-low-leakage Stop mode.

28.5.5 enum smc_stop_submode_t

Enumerator

kSMC_StopSub0 Stop submode 0, for VLLS0/LLS0.
kSMC_StopSub1 Stop submode 1, for VLLS1/LLS1.
kSMC_StopSub2 Stop submode 2, for VLLS2/LLS2.
kSMC_StopSub3 Stop submode 3, for VLLS3/LLS3.

28.5.6 enum smc_partial_stop_option_t

Enumerator

kSMC_PartialStop STOP - Normal Stop mode.
kSMC_PartialStop1 Partial Stop with both system and bus clocks disabled.
kSMC_PartialStop2 Partial Stop with system clock disabled and bus clock enabled.

28.5.7 enum _smc_status

Enumerator

kStatus_SMC_StopAbort Entering Stop mode is abort.

28.6 Function Documentation

28.6.1 static void SMC_SetPowerModeProtection (SMC_Type ∗ *base,* uint8_t *allowedModes*) `[inline], [static]`

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the smc_power_mode_protection_t. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map. For example, to allow LLS and VLLS, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeVlls | kSMC_AllowPowerModeVlps). To allow all modes, use SMC_SetPowerModeProtection(kSMC_AllowPowerModeAll).

Parameters

base	SMC peripheral base address.
allowedModes	Bitmap of the allowed power modes.

28.6.2 static smc_power_state_t SMC_GetPowerModeState (SMC_Type * *base*) [inline], [static]

This function returns the current power mode status. After the application switches the power mode, it should always check the status to check whether it runs into the specified mode or not. The application should check this mode before switching to a different mode. The system requires that only certain modes can switch to other specific modes. See the reference manual for details and the smc_power_state_t for information about the power status.

Parameters

base	SMC peripheral base address.

Returns

Current power mode status.

28.6.3 void SMC_PreEnterStopModes (void)

This function should be called before entering STOP/VLPS/LLS/VLLS modes.

28.6.4 void SMC_PostExitStopModes (void)

This function should be called after wake up from STOP/VLPS/LLS/VLLS modes. It is used with SMC_PreEnterStopModes.

28.6.5 static void SMC_PreEnterWaitModes (void) [inline], [static]

This function should be called before entering WAIT/VLPW modes.

28.6.6 static void SMC_PostExitWaitModes (void) [inline], [static]

This function should be called after wake up from WAIT/VLPW modes. It is used with SMC_PreEnterWaitModes.

28.6.7 status_t SMC_SetPowerModeRun (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

28.6.8 status_t SMC_SetPowerModeWait (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

28.6.9 status_t SMC_SetPowerModeStop (SMC_Type * *base*, smc_partial_stop_option_t *option*)

Parameters

base	SMC peripheral base address.
option	Partial Stop mode option.

Returns

SMC configuration error code.

28.6.10 status_t SMC_SetPowerModeVlpr (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

28.6.11 status_t SMC_SetPowerModeVlpw (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

28.6.12 status_t SMC_SetPowerModeVlps (SMC_Type * *base*)

Parameters

base	SMC peripheral base address.

Returns

SMC configuration error code.

28.6.13 status_t SMC_SetPowerModeLls (SMC_Type * *base,* const smc_power_mode_lls_config_t * *config*)

Parameters

base	SMC peripheral base address.

config	The LLS power mode configuration structure

Returns

SMC configuration error code.

28.6.14 status_t SMC_SetPowerModeVlls (SMC_Type * *base,* const smc_power_mode_vlls_config_t * *config*)

Parameters

base	SMC peripheral base address.
config	The VLLS power mode configuration structure.

Returns

SMC configuration error code.

Chapter 29
TPM: Timer PWM Module

29.1 Overview

The MCUXpresso SDK provides a driver for the Timer PWM Module (TPM) of MCUXpresso SDK devices.

The TPM driver supports the generation of PWM signals, input capture, and output compare modes. On some SoCs, the driver supports the generation of combined PWM signals, dual-edge capture, and quadrature decoder modes. The driver also supports configuring each of the TPM fault inputs. The fault input is available only on some SoCs.

The function TPM_Init() initializes the TPM with a specified configurations. The function TPM_GetDefaultConfig() gets the default configurations. On some SoCs, the initialization function issues a software reset to reset the TPM internal logic. The initialization function configures the TPM's behavior when it receives a trigger input and its operation in doze and debug modes.

The function TPM_Deinit() disables the TPM counter and turns off the module clock.

The function TPM_SetupPwm() sets up TPM channels for the PWM output. The function can set up the PWM signal properties for multiple channels. Each channel has its own tpm_chnl_pwm_signal_param_t structure that is used to specify the output signals duty cycle and level-mode. However, the same PWM period and PWM mode is applied to all channels requesting a PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 where 0=inactive signal (0% duty cycle) and 100=always active signal (100% duty cycle). When generating a combined PWM signal, the channel number passed refers to a channel pair number, for example 0 refers to channel 0 and 1, 1 refers to channels 2 and 3.

The function TPM_UpdatePwmDutycycle() updates the PWM signal duty cycle of a particular TPM channel.

The function TPM_UpdateChnlEdgeLevelSelect() updates the level select bits of a particular TPM channel. This can be used to disable the PWM output when making changes to the PWM signal.

The function TPM_SetupInputCapture() sets up a TPM channel for input capture. The user can specify the capture edge.

The function TPM_SetupDualEdgeCapture() can be used to measure the pulse width of a signal. This is available only for certain SoCs. A channel pair is used during the capture with the input signal coming through a channel that can be configured. The user can specify the capture edge for each channel and any filter value to be used when processing the input signal.

The function TPM_SetupOutputCompare() sets up a TPM channel for output comparison. The user can specify the channel output on a successful comparison and a comparison value.

The function TPM_SetupQuadDecode() sets up TPM channels 0 and 1 for quad decode, which is available only for certain SoCs. The user can specify the quad decode mode, polarity, and filter properties for each

input signal.

The function TPM_SetupFault() sets up the properties for each fault, which is available only for certain SoCs. The user can specify the fault polarity and whether to use a filter on a fault input. The overall fault filter value and fault control mode are set up during initialization.

Provides functions to get and clear the TPM status.

Provides functions to enable/disable TPM interrupts and get current enabled interrupts.

29.2 Typical use case

29.2.1 PWM output

Output the PWM signal on 2 TPM channels with different duty cycles. Periodically update the PWM signal duty cycle.

```
int main(void)
{
    bool brightnessUp = true; /* Indicates whether the LED is brighter or dimmer. */
    tpm_config_t tpmInfo;
    uint8_t updatedDutycycle = 0U;
    tpm_chnl_pwm_signal_param_t tpmParam[2];

    /* Configures the TPM parameters with frequency 24 kHZ. */
    tpmParam[0].chnlNumber = (tpm_chnl_t)BOARD_FIRST_TPM_CHANNEL;
    tpmParam[0].level = kTPM_LowTrue;
    tpmParam[0].dutyCyclePercent = 0U;

    tpmParam[1].chnlNumber = (tpm_chnl_t)BOARD_SECOND_TPM_CHANNEL;
    tpmParam[1].level = kTPM_LowTrue;
    tpmParam[1].dutyCyclePercent = 0U;

    /* Board pin, clock, and debug console initialization. */
    BOARD_InitHardware();

    TPM_GetDefaultConfig(&tpmInfo);
    /* Initializes the TPM module. */
    TPM_Init(BOARD_TPM_BASEADDR, &tpmInfo);

    TPM_SetupPwm(BOARD_TPM_BASEADDR, tpmParam, 2U,
      kTPM_EdgeAlignedPwm, 24000U, TPM_SOURCE_CLOCK);
    TPM_StartTimer(BOARD_TPM_BASEADDR, kTPM_SystemClock);
    while (1)
    {
        /* Delays to see the change of LED brightness. */
        delay();

        if (brightnessUp)
        {
            /* Increases a duty cycle until it reaches a limited value. */
            if (++updatedDutycycle == 100U)
            {
                brightnessUp = false;
            }
        }
        else
        {
            /* Decreases a duty cycle until it reaches a limited value. */
            if (--updatedDutycycle == 0U)
            {
                brightnessUp = true;
```

```
            }
        }
        /* Starts PWM mode with an updated duty cycle. */
        TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR, (
      tpm_chnl_t)BOARD_FIRST_TPM_CHANNEL, kTPM_EdgeAlignedPwm,
                                updatedDutycycle);
        TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR, (
      tpm_chnl_t)BOARD_SECOND_TPM_CHANNEL, kTPM_EdgeAlignedPwm,
                                updatedDutycycle);
    }
}
```

Data Structures

- struct tpm_chnl_pwm_signal_param_t
  Options to configure a TPM channel's PWM signal. More...
- struct tpm_dual_edge_capture_param_t
  TPM dual edge capture parameters. More...
- struct tpm_phase_params_t
  TPM quadrature decode phase parameters. More...
- struct tpm_config_t
  TPM config structure. More...

Enumerations

- enum tpm_chnl_t {
  kTPM_Chnl_0 = 0U,
  kTPM_Chnl_1,
  kTPM_Chnl_2,
  kTPM_Chnl_3,
  kTPM_Chnl_4,
  kTPM_Chnl_5,
  kTPM_Chnl_6,
  kTPM_Chnl_7 }
  List of TPM channels.
- enum tpm_pwm_mode_t {
  kTPM_EdgeAlignedPwm = 0U,
  kTPM_CenterAlignedPwm,
  kTPM_CombinedPwm }
  TPM PWM operation modes.
- enum tpm_pwm_level_select_t {
  kTPM_NoPwmSignal = 0U,
  kTPM_LowTrue,
  kTPM_HighTrue }
  TPM PWM output pulse mode: high-true, low-true or no output.
- enum tpm_trigger_select_t
  Trigger options available.
- enum tpm_trigger_source_t {
  kTPM_TriggerSource_External = 0U,
  kTPM_TriggerSource_Internal }
  Trigger source options available.

- enum tpm_output_compare_mode_t {
  kTPM_NoOutputSignal = (1U << TPM_CnSC_MSA_SHIFT),
  kTPM_ToggleOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_ClearOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_SetOnMatch = ((1U << TPM_CnSC_MSA_SHIFT) | (3U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_HighPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (1U << TPM_CnSC_ELSA_SHIFT)),
  kTPM_LowPulseOutput = ((3U << TPM_CnSC_MSA_SHIFT) | (2U << TPM_CnSC_ELSA_SHIFT)) }
  TPM output compare modes.
- enum tpm_input_capture_edge_t {
  kTPM_RisingEdge = (1U << TPM_CnSC_ELSA_SHIFT),
  kTPM_FallingEdge = (2U << TPM_CnSC_ELSA_SHIFT),
  kTPM_RiseAndFallEdge = (3U << TPM_CnSC_ELSA_SHIFT) }
  TPM input capture edge.
- enum tpm_quad_decode_mode_t {
  kTPM_QuadPhaseEncode = 0U,
  kTPM_QuadCountAndDir }
  TPM quadrature decode modes.
- enum tpm_phase_polarity_t {
  kTPM_QuadPhaseNormal = 0U,
  kTPM_QuadPhaseInvert }
  TPM quadrature phase polarities.
- enum tpm_clock_source_t {
  kTPM_SystemClock = 1U,
  kTPM_ExternalClock }
  TPM clock source selection.
- enum tpm_clock_prescale_t {
  kTPM_Prescale_Divide_1 = 0U,
  kTPM_Prescale_Divide_2,
  kTPM_Prescale_Divide_4,
  kTPM_Prescale_Divide_8,
  kTPM_Prescale_Divide_16,
  kTPM_Prescale_Divide_32,
  kTPM_Prescale_Divide_64,
  kTPM_Prescale_Divide_128 }
  TPM prescale value selection for the clock source.
- enum tpm_interrupt_enable_t {

kTPM_Chnl0InterruptEnable = (1U << 0),
kTPM_Chnl1InterruptEnable = (1U << 1),
kTPM_Chnl2InterruptEnable = (1U << 2),
kTPM_Chnl3InterruptEnable = (1U << 3),
kTPM_Chnl4InterruptEnable = (1U << 4),
kTPM_Chnl5InterruptEnable = (1U << 5),
kTPM_Chnl6InterruptEnable = (1U << 6),
kTPM_Chnl7InterruptEnable = (1U << 7),
kTPM_TimeOverflowInterruptEnable = (1U << 8) }
List of TPM interrupts.

- enum tpm_status_flags_t {
  kTPM_Chnl0Flag = (1U << 0),
  kTPM_Chnl1Flag = (1U << 1),
  kTPM_Chnl2Flag = (1U << 2),
  kTPM_Chnl3Flag = (1U << 3),
  kTPM_Chnl4Flag = (1U << 4),
  kTPM_Chnl5Flag = (1U << 5),
  kTPM_Chnl6Flag = (1U << 6),
  kTPM_Chnl7Flag = (1U << 7),
  kTPM_TimeOverflowFlag = (1U << 8) }
  List of TPM flags.

Driver version

- #define FSL_TPM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))
  Version 2.0.2.

Initialization and deinitialization

- void TPM_Init (TPM_Type *base, const tpm_config_t *config)
  Ungates the TPM clock and configures the peripheral for basic operation.
- void TPM_Deinit (TPM_Type *base)
  Stops the counter and gates the TPM clock.
- void TPM_GetDefaultConfig (tpm_config_t *config)
  Fill in the TPM config struct with the default settings.

Channel mode operations

- status_t TPM_SetupPwm (TPM_Type *base, const tpm_chnl_pwm_signal_param_t *chnlParams, uint8_t numOfChnls, tpm_pwm_mode_t mode, uint32_t pwmFreq_Hz, uint32_t srcClock_Hz)
  Configures the PWM signal parameters.
- void TPM_UpdatePwmDutycycle (TPM_Type *base, tpm_chnl_t chnlNumber, tpm_pwm_mode_t currentPwmMode, uint8_t dutyCyclePercent)
  Update the duty cycle of an active PWM signal.
- void TPM_UpdateChnlEdgeLevelSelect (TPM_Type *base, tpm_chnl_t chnlNumber, uint8_t level)
  Update the edge level selection for a channel.
- void TPM_SetupInputCapture (TPM_Type *base, tpm_chnl_t chnlNumber, tpm_input_capture_-edge_t captureMode)

Enables capturing an input signal on the channel using the function parameters.

- void TPM_SetupOutputCompare (TPM_Type *base, tpm_chnl_t chnlNumber, tpm_output_compare_mode_t compareMode, uint32_t compareValue)
  Configures the TPM to generate timed pulses.
- void TPM_SetupDualEdgeCapture (TPM_Type *base, tpm_chnl_t chnlPairNumber, const tpm_dual_edge_capture_param_t *edgeParam, uint32_t filterValue)
  Configures the dual edge capture mode of the TPM.
- void TPM_SetupQuadDecode (TPM_Type *base, const tpm_phase_params_t *phaseAParams, const tpm_phase_params_t *phaseBParams, tpm_quad_decode_mode_t quadMode)
  Configures the parameters and activates the quadrature decode mode.

Interrupt Interface

- void TPM_EnableInterrupts (TPM_Type *base, uint32_t mask)
  Enables the selected TPM interrupts.
- void TPM_DisableInterrupts (TPM_Type *base, uint32_t mask)
  Disables the selected TPM interrupts.
- uint32_t TPM_GetEnabledInterrupts (TPM_Type *base)
  Gets the enabled TPM interrupts.

Status Interface

- static uint32_t TPM_GetStatusFlags (TPM_Type *base)
  Gets the TPM status flags.
- static void TPM_ClearStatusFlags (TPM_Type *base, uint32_t mask)
  Clears the TPM status flags.

Read and write the timer period

- static void TPM_SetTimerPeriod (TPM_Type *base, uint32_t ticks)
  Sets the timer period in units of ticks.
- static uint32_t TPM_GetCurrentTimerCount (TPM_Type *base)
  Reads the current timer counting value.

Timer Start and Stop

- static void TPM_StartTimer (TPM_Type *base, tpm_clock_source_t clockSource)
  Starts the TPM counter.
- static void TPM_StopTimer (TPM_Type *base)
  Stops the TPM counter.

29.3 Data Structure Documentation

29.3.1 struct tpm_chnl_pwm_signal_param_t

Data Fields

- tpm_chnl_t chnlNumber
  TPM channel to configure.

- tpm_pwm_level_select_t level
  PWM output active level select.
- uint8_t dutyCyclePercent
  PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...
- uint8_t firstEdgeDelayPercent
  Used only in combined PWM mode to generate asymmetrical PWM.

29.3.1.0.5.5 Field Documentation

29.3.1.0.5.5.1 tpm_chnl_t tpm_chnl_pwm_signal_param_t::chnlNumber

In combined mode (available in some SoC's, this represents the channel pair number

29.3.1.0.5.5.2 uint8_t tpm_chnl_pwm_signal_param_t::dutyCyclePercent

100=always active signal (100% duty cycle)

29.3.1.0.5.5.3 uint8_t tpm_chnl_pwm_signal_param_t::firstEdgeDelayPercent

Specifies the delay to the first edge in a PWM period. If unsure, leave as 0; Should be specified as percentage of the PWM period

29.3.2 struct tpm_dual_edge_capture_param_t

Note

This mode is available only on some SoC's.

Data Fields

- bool enableSwap
  true: Use channel n+1 input, channel n input is ignored; false: Use channel n input, channel n+1 input is ignored
- tpm_input_capture_edge_t currChanEdgeMode
  Input capture edge select for channel n.
- tpm_input_capture_edge_t nextChanEdgeMode
  Input capture edge select for channel n+1.

29.3.3 struct tpm_phase_params_t

Data Fields

- uint32_t phaseFilterVal
  Filter value, filter is disabled when the value is zero.
- tpm_phase_polarity_t phasePolarity
  Phase polarity.

29.3.4 struct tpm_config_t

This structure holds the configuration settings for the TPM peripheral. To initialize this structure to reasonable defaults, call the TPM_GetDefaultConfig() function and pass a pointer to your config structure instance.

The config struct can be made const so it resides in flash

Data Fields

- tpm_clock_prescale_t prescale
  Select TPM clock prescale value.
- bool useGlobalTimeBase
  true: Use of an external global time base is enabled; false: disabled
- tpm_trigger_select_t triggerSelect
  Input trigger to use for controlling the counter operation.
- tpm_trigger_source_t triggerSource
  Decides if we use external or internal trigger.
- bool enableDoze
  true: TPM counter is paused in doze mode; false: TPM counter continues in doze mode
- bool enableDebugMode
  true: TPM counter continues in debug mode; false: TPM counter is paused in debug mode
- bool enableReloadOnTrigger
  true: TPM counter is reloaded on trigger; false: TPM counter not reloaded
- bool enableStopOnOverflow
  true: TPM counter stops after overflow; false: TPM counter continues running after overflow
- bool enableStartOnTrigger
  true: TPM counter only starts when a trigger is detected; false: TPM counter starts immediately
- bool enablePauseOnTrigger
  true: TPM counter will pause while trigger remains asserted; false: TPM counter continues running

29.3.4.0.5.6 Field Documentation

29.3.4.0.5.6.1 tpm_trigger_source_t tpm_config_t::triggerSource

29.4 Enumeration Type Documentation

29.4.1 enum tpm_chnl_t

Note

Actual number of available channels is SoC dependent

Enumerator

kTPM_Chnl_0 TPM channel number 0.
kTPM_Chnl_1 TPM channel number 1.
kTPM_Chnl_2 TPM channel number 2.
kTPM_Chnl_3 TPM channel number 3.

kTPM_Chnl_4 TPM channel number 4.
kTPM_Chnl_5 TPM channel number 5.
kTPM_Chnl_6 TPM channel number 6.
kTPM_Chnl_7 TPM channel number 7.

29.4.2 enum tpm_pwm_mode_t

Enumerator

kTPM_EdgeAlignedPwm Edge aligned PWM.
kTPM_CenterAlignedPwm Center aligned PWM.
kTPM_CombinedPwm Combined PWM.

29.4.3 enum tpm_pwm_level_select_t

Enumerator

kTPM_NoPwmSignal No PWM output on pin.
kTPM_LowTrue Low true pulses.
kTPM_HighTrue High true pulses.

29.4.4 enum tpm_trigger_select_t

This is used for both internal & external trigger sources (external option available in certain SoC's)

Note

The actual trigger options available is SoC-specific.

29.4.5 enum tpm_trigger_source_t

Note

This selection is available only on some SoC's. For SoC's without this selection, the only trigger source available is internal triger.

Enumerator

kTPM_TriggerSource_External Use external trigger input.
kTPM_TriggerSource_Internal Use internal trigger.

29.4.6 enum tpm_output_compare_mode_t

Enumerator

kTPM_NoOutputSignal No channel output when counter reaches CnV.
kTPM_ToggleOnMatch Toggle output.
kTPM_ClearOnMatch Clear output.
kTPM_SetOnMatch Set output.
kTPM_HighPulseOutput Pulse output high.
kTPM_LowPulseOutput Pulse output low.

29.4.7 enum tpm_input_capture_edge_t

Enumerator

kTPM_RisingEdge Capture on rising edge only.
kTPM_FallingEdge Capture on falling edge only.
kTPM_RiseAndFallEdge Capture on rising or falling edge.

29.4.8 enum tpm_quad_decode_mode_t

Note

This mode is available only on some SoC's.

Enumerator

kTPM_QuadPhaseEncode Phase A and Phase B encoding mode.
kTPM_QuadCountAndDir Count and direction encoding mode.

29.4.9 enum tpm_phase_polarity_t

Enumerator

kTPM_QuadPhaseNormal Phase input signal is not inverted.
kTPM_QuadPhaseInvert Phase input signal is inverted.

29.4.10 enum tpm_clock_source_t

Enumerator

kTPM_SystemClock System clock.
kTPM_ExternalClock External clock.

29.4.11 enum tpm_clock_prescale_t

Enumerator

kTPM_Prescale_Divide_1 Divide by 1.
kTPM_Prescale_Divide_2 Divide by 2.
kTPM_Prescale_Divide_4 Divide by 4.
kTPM_Prescale_Divide_8 Divide by 8.
kTPM_Prescale_Divide_16 Divide by 16.
kTPM_Prescale_Divide_32 Divide by 32.
kTPM_Prescale_Divide_64 Divide by 64.
kTPM_Prescale_Divide_128 Divide by 128.

29.4.12 enum tpm_interrupt_enable_t

Enumerator

kTPM_Chnl0InterruptEnable Channel 0 interrupt.
kTPM_Chnl1InterruptEnable Channel 1 interrupt.
kTPM_Chnl2InterruptEnable Channel 2 interrupt.
kTPM_Chnl3InterruptEnable Channel 3 interrupt.
kTPM_Chnl4InterruptEnable Channel 4 interrupt.
kTPM_Chnl5InterruptEnable Channel 5 interrupt.
kTPM_Chnl6InterruptEnable Channel 6 interrupt.
kTPM_Chnl7InterruptEnable Channel 7 interrupt.
kTPM_TimeOverflowInterruptEnable Time overflow interrupt.

29.4.13 enum tpm_status_flags_t

Enumerator

kTPM_Chnl0Flag Channel 0 flag.
kTPM_Chnl1Flag Channel 1 flag.
kTPM_Chnl2Flag Channel 2 flag.
kTPM_Chnl3Flag Channel 3 flag.
kTPM_Chnl4Flag Channel 4 flag.
kTPM_Chnl5Flag Channel 5 flag.
kTPM_Chnl6Flag Channel 6 flag.
kTPM_Chnl7Flag Channel 7 flag.
kTPM_TimeOverflowFlag Time overflow flag.

29.5 Function Documentation

29.5.1 void TPM_Init (TPM_Type ∗ *base,* const tpm_config_t ∗ *config*)

Note

This API should be called at the beginning of the application using the TPM driver.

Parameters

base	TPM peripheral base address
config	Pointer to user's TPM config structure.

29.5.2 void TPM_Deinit (TPM_Type * *base*)

Parameters

base	TPM peripheral base address

29.5.3 void TPM_GetDefaultConfig (tpm_config_t * *config*)

The default values are:

```
*     config->prescale = kTPM_Prescale_Divide_1;
*     config->useGlobalTimeBase = false;
*     config->dozeEnable = false;
*     config->dbgMode = false;
*     config->enableReloadOnTrigger = false;
*     config->enableStopOnOverflow = false;
*     config->enableStartOnTrigger = false;
*#if FSL_FEATURE_TPM_HAS_PAUSE_COUNTER_ON_TRIGGER
*     config->enablePauseOnTrigger = false;
*#endif
*     config->triggerSelect = kTPM_Trigger_Select_0;
*#if FSL_FEATURE_TPM_HAS_EXTERNAL_TRIGGER_SELECTION
*     config->triggerSource = kTPM_TriggerSource_External;
*#endif
*
```

Parameters

config	Pointer to user's TPM config structure.

29.5.4 status_t TPM_SetupPwm (TPM_Type * *base*, const tpm_chnl_pwm_signal_param_t * *chnlParams*, uint8_t *numOfChnls*, tpm_pwm_mode_t *mode*, uint32_t *pwmFreq_Hz*, uint32_t *srcClock_Hz*)

User calls this function to configure the PWM signals period, mode, dutycycle and edge. Use this function to configure all the TPM channels that will be used to output a PWM signal

Parameters

base	TPM peripheral base address
chnlParams	Array of PWM channel parameters to configure the channel(s)
numOfChnls	Number of channels to configure, this should be the size of the array passed in
mode	PWM operation mode, options available in enumeration tpm_pwm_mode_t
pwmFreq_Hz	PWM signal frequency in Hz
srcClock_Hz	TPM counter clock in Hz

Returns

kStatus_Success if the PWM setup was successful, kStatus_Error on failure

29.5.5 void TPM_UpdatePwmDutycycle (TPM_Type * *base*, tpm_chnl_t *chnlNumber*, tpm_pwm_mode_t *currentPwmMode*, uint8_t *dutyCyclePercent*)

Parameters

base	TPM peripheral base address
chnlNumber	The channel number. In combined mode, this represents the channel pair number
currentPwm-Mode	The current PWM mode set during PWM setup
dutyCycle-Percent	New PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)... 100=active signal (100% duty cycle)

29.5.6 void TPM_UpdateChnlEdgeLevelSelect (TPM_Type * *base*, tpm_chnl_t *chnlNumber*, uint8_t *level*)

Parameters

base	TPM peripheral base address

chnlNumber	The channel number
level	The level to be set to the ELSnB:ELSnA field; valid values are 00, 01, 10, 11. See the appropriate SoC reference manual for details about this field.

29.5.7 void TPM_SetupInputCapture (TPM_Type * *base*, tpm_chnl_t *chnlNumber*, tpm_input_capture_edge_t *captureMode*)

When the edge specified in the captureMode argument occurs on the channel, the TPM counter is captured into the CnV register. The user has to read the CnV register separately to get this value.

Parameters

base	TPM peripheral base address
chnlNumber	The channel number
captureMode	Specifies which edge to capture

29.5.8 void TPM_SetupOutputCompare (TPM_Type * *base*, tpm_chnl_t *chnlNumber*, tpm_output_compare_mode_t *compareMode*, uint32_t *compareValue*)

When the TPM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

base	TPM peripheral base address
chnlNumber	The channel number
compareMode	Action to take on the channel output when the compare condition is met
compareValue	Value to be programmed in the CnV register.

29.5.9 void TPM_SetupDualEdgeCapture (TPM_Type * *base*, tpm_chnl_t *chnlPairNumber*, const tpm_dual_edge_capture_param_t * *edgeParam*, uint32_t *filterValue*)

This function allows to measure a pulse width of the signal on the input of channel of a channel pair. The filter function is disabled if the filterVal argument passed is zero.

Parameters

base	TPM peripheral base address
chnlPair-Number	The TPM channel pair number; options are 0, 1, 2, 3
edgeParam	Sets up the dual edge capture function
filterValue	Filter value, specify 0 to disable filter.

29.5.10 void TPM_SetupQuadDecode (TPM_Type ∗ *base,* const tpm_phase_params_t ∗ *phaseAParams,* const tpm_phase_params_t ∗ *phaseBParams,* tpm_quad_decode_mode_t *quadMode*)

Parameters

base	TPM peripheral base address
phaseAParams	Phase A configuration parameters
phaseBParams	Phase B configuration parameters
quadMode	Selects encoding mode used in quadrature decoder mode

29.5.11 void TPM_EnableInterrupts (TPM_Type ∗ *base,* uint32_t *mask*)

Parameters

base	TPM peripheral base address
mask	The interrupts to enable. This is a logical OR of members of the enumeration tpm_-interrupt_enable_t

29.5.12 void TPM_DisableInterrupts (TPM_Type ∗ *base,* uint32_t *mask*)

Parameters

base	TPM peripheral base address
mask	The interrupts to disable. This is a logical OR of members of the enumeration tpm_interrupt_enable_t

29.5.13 uint32_t TPM_GetEnabledInterrupts (TPM_Type * *base*)

Parameters

base	TPM peripheral base address

Returns

The enabled interrupts. This is the logical OR of members of the enumeration tpm_interrupt_enable_t

29.5.14 static uint32_t TPM_GetStatusFlags (TPM_Type * *base*) [inline], [static]

Parameters

base	TPM peripheral base address

Returns

The status flags. This is the logical OR of members of the enumeration tpm_status_flags_t

29.5.15 static void TPM_ClearStatusFlags (TPM_Type * *base*, uint32_t *mask*) [inline], [static]

Parameters

base	TPM peripheral base address

mask	The status flags to clear. This is a logical OR of members of the enumeration tpm_status_flags_t

29.5.16 static void TPM_SetTimerPeriod (TPM_Type * *base,* uint32_t *ticks*) [inline], [static]

Timers counts from 0 until it equals the count value set here. The count value is written to the MOD register.

Note

1. This API allows the user to use the TPM module as a timer. Do not mix usage of this API with TPM's PWM setup API's.
2. Call the utility macros provided in the fsl_common.h to convert usec or msec to ticks.

Parameters

base	TPM peripheral base address
ticks	A timer period in units of ticks, which should be equal or greater than 1.

29.5.17 static uint32_t TPM_GetCurrentTimerCount (TPM_Type * *base*) [inline], [static]

This function returns the real-time timer counting value in a range from 0 to a timer period.

Note

Call the utility macros provided in the fsl_common.h to convert ticks to usec or msec.

Parameters

base	TPM peripheral base address

Returns

The current counter value in ticks

29.5.18 static void TPM_StartTimer (TPM_Type * *base,* tpm_clock_source_t *clockSource*) [inline], [static]

Parameters

base	TPM peripheral base address
clockSource	TPM clock source; once clock source is set the counter will start running

29.5.19 static void TPM_StopTimer (TPM_Type * *base*) [inline], [static]

Parameters

base	TPM peripheral base address

Chapter 30
TRNG: True Random Number Generator

30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the True Random Number Generator (TRNG) module of MCUXpresso SDK devices.

The True Random Number Generator is a hardware accelerator module that generates a 512-bit entropy as needed by an entropy consuming module or by other post processing functions. A typical entropy consumer is a pseudo random number generator (PRNG) which can be implemented to achieve both true randomness and cryptographic strength random numbers using the TRNG output as its entropy seed. The entropy generated by a TRNG is intended for direct use by functions that generate secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

30.2 TRNG Initialization

1. Define the TRNG user configuration structure. Use TRNG_InitUserConfigDefault() function to set it to default TRNG configuration values.
2. Initialize the TRNG module, call the TRNG_Init() function, and pass the user configuration structure. This function automatically enables the TRNG module and its clock. After that, the TRNG is enabled and the entropy generation starts working.
3. To disable the TRNG module, call the TRNG_Deinit() function.

30.3 Get random data from TRNG

1. TRNG_GetRandomData() function gets random data from the TRNG module.

This example code shows how to initialize and get random data from the TRNG driver.

```
{
    trng_user_config_t  trngConfig;
    status_t            status;
    uint32_t            data;

    /* Initialize TRNG configuration structure to default.*/
    TRNG_InitUserConfigDefault(&trngConfig);

    /* Initialize TRNG */
    status = TRNG_Init(TRNG0, &trngConfig);

    if (status == kStatus_Success)
    {
        /* Read Random data*/
        if((status = TRNG_GetRandomData(TRNG0, data, sizeof(data))) ==
      kStatus_TRNG_Success)
        {
            /* Print data*/
            PRINTF("Random = 0x%X\r\n", i, data );

            PRINTF("Succeed.\r\n");
```

```
        }
        else
        {
            PRINTF("TRNG failed! (0x%x)\r\n", status);
        }

        /* Deinitialize TRNG*/
        TRNG_Deinit(TRNG0);
    }
    else
    {
        PRINTF("TRNG initialization failed!\r\n");
    }
}
```

Data Structures

- struct trng_statistical_check_limit_t
  Data structure for definition of statistical check limits. More...
- struct trng_config_t
  Data structure for the TRNG initialization. More...

Enumerations

- enum trng_sample_mode_t {
  kTRNG_SampleModeVonNeumann = 0U,
  kTRNG_SampleModeRaw = 1U,
  kTRNG_SampleModeVonNeumannRaw }
  TRNG sample mode.
- enum trng_clock_mode_t {
  kTRNG_ClockModeRingOscillator = 0U,
  kTRNG_ClockModeSystem = 1U }
  TRNG clock mode.
- enum trng_ring_osc_div_t {
  kTRNG_RingOscDiv0 = 0U,
  kTRNG_RingOscDiv2 = 1U,
  kTRNG_RingOscDiv4 = 2U,
  kTRNG_RingOscDiv8 = 3U }
  TRNG ring oscillator divide.

Functions

- status_t TRNG_GetDefaultConfig (trng_config_t ∗userConfig)
  Initializes the user configuration structure to default values.
- status_t TRNG_Init (TRNG_Type ∗base, const trng_config_t ∗userConfig)
  Initializes the TRNG.
- void TRNG_Deinit (TRNG_Type ∗base)
  Shuts down the TRNG.
- status_t TRNG_GetRandomData (TRNG_Type ∗base, void ∗data, size_t dataSize)
  Gets random data.

Driver version

- #define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

TRNG driver version 2.0.1.

30.4 Data Structure Documentation

30.4.1 struct trng_statistical_check_limit_t

Used by trng_config_t.

Data Fields

- uint32_t maximum
  Maximum limit.
- uint32_t minimum
  Minimum limit.

30.4.1.0.5.7 Field Documentation

30.4.1.0.5.7.1 uint32_t trng_statistical_check_limit_t::maximum

30.4.1.0.5.7.2 uint32_t trng_statistical_check_limit_t::minimum

30.4.2 struct trng_config_t

This structure initializes the TRNG by calling the the TRNG_Init() function. It contains all TRNG configurations.

Data Fields

- bool lock
  Disable programmability of TRNG registers.
- trng_clock_mode_t clockMode
  Clock mode used to operate TRNG.
- trng_ring_osc_div_t ringOscDiv
  Ring oscillator divide used by TRNG.
- trng_sample_mode_t sampleMode
  Sample mode of the TRNG ring oscillator.
- uint16_t entropyDelay
  Entropy Delay.
- uint16_t sampleSize
  Sample Size.
- uint16_t sparseBitLimit
  Sparse Bit Limit which defines the maximum number of consecutive samples that may be discarded before an error is generated.
- uint8_t retryCount
  Retry count.
- uint8_t longRunMaxLimit

Largest allowable number of consecutive samples of all 1, or all 0, that is allowed during the Entropy generation.

- trng_statistical_check_limit_t monobitLimit
  Maximum and minimum limits for statistical check of number of ones/zero detected during entropy generation.
- trng_statistical_check_limit_t runBit1Limit
  Maximum and minimum limits for statistical check of number of runs of length 1 detected during entropy generation.
- trng_statistical_check_limit_t runBit2Limit
  Maximum and minimum limits for statistical check of number of runs of length 2 detected during entropy generation.
- trng_statistical_check_limit_t runBit3Limit
  Maximum and minimum limits for statistical check of number of runs of length 3 detected during entropy generation.
- trng_statistical_check_limit_t runBit4Limit
  Maximum and minimum limits for statistical check of number of runs of length 4 detected during entropy generation.
- trng_statistical_check_limit_t runBit5Limit
  Maximum and minimum limits for statistical check of number of runs of length 5 detected during entropy generation.
- trng_statistical_check_limit_t runBit6PlusLimit
  Maximum and minimum limits for statistical check of number of runs of length 6 or more detected during entropy generation.
- trng_statistical_check_limit_t pokerLimit
  Maximum and minimum limits for statistical check of "Poker Test".
- trng_statistical_check_limit_t frequencyCountLimit
  Maximum and minimum limits for statistical check of entropy sample frequency count.

30.4.2.0.5.8 Field Documentation

30.4.2.0.5.8.1 bool trng_config_t::lock

30.4.2.0.5.8.2 trng_clock_mode_t trng_config_t::clockMode

30.4.2.0.5.8.3 trng_ring_osc_div_t trng_config_t::ringOscDiv

30.4.2.0.5.8.4 trng_sample_mode_t trng_config_t::sampleMode

30.4.2.0.5.8.5 uint16_t trng_config_t::entropyDelay

Defines the length (in system clocks) of each Entropy sample taken.

30.4.2.0.5.8.6 uint16_t trng_config_t::sampleSize

Defines the total number of Entropy samples that will be taken during Entropy generation.

30.4.2.0.5.8.7 uint16_t trng_config_t::sparseBitLimit

This limit is used only for during von Neumann sampling (enabled by TRNG_HAL_SetSampleMode()). Samples are discarded if two consecutive raw samples are both 0 or both 1. If this discarding occurs for a

long period of time, it indicates that there is insufficient Entropy.

30.4.2.0.5.8.8 uint8_t trng_config_t::retryCount

It defines the number of times a statistical check may fails during the TRNG Entropy Generation before generating an error.

30.4.2.0.5.8.9 uint8_t trng_config_t::longRunMaxLimit

30.4.2.0.5.8.10 trng_statistical_check_limit_t trng_config_t::monobitLimit

30.4.2.0.5.8.11 trng_statistical_check_limit_t trng_config_t::runBit1Limit

30.4.2.0.5.8.12 trng_statistical_check_limit_t trng_config_t::runBit2Limit

30.4.2.0.5.8.13 trng_statistical_check_limit_t trng_config_t::runBit3Limit

30.4.2.0.5.8.14 trng_statistical_check_limit_t trng_config_t::runBit4Limit

30.4.2.0.5.8.15 trng_statistical_check_limit_t trng_config_t::runBit5Limit

30.4.2.0.5.8.16 trng_statistical_check_limit_t trng_config_t::runBit6PlusLimit

30.4.2.0.5.8.17 trng_statistical_check_limit_t trng_config_t::pokerLimit

30.4.2.0.5.8.18 trng_statistical_check_limit_t trng_config_t::frequencyCountLimit

30.5 Macro Definition Documentation

30.5.1 #define FSL_TRNG_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))

Current version: 2.0.1

Change log:

- Version 2.0.1
  - add support for KL8x and KL28Z
  - update default OSCDIV for K81 to divide by 2

30.6 Enumeration Type Documentation

30.6.1 enum trng_sample_mode_t

Used by trng_config_t.

Enumerator

kTRNG_SampleModeVonNeumann Use von Neumann data in both Entropy shifter and Statistical Checker.

kTRNG_SampleModeRaw Use raw data into both Entropy shifter and Statistical Checker.

kTRNG_SampleModeVonNeumannRaw Use von Neumann data in Entropy shifter. Use raw data into Statistical Checker.

30.6.2 enum trng_clock_mode_t

Used by trng_config_t.

Enumerator

kTRNG_ClockModeRingOscillator Ring oscillator is used to operate the TRNG (default).
kTRNG_ClockModeSystem System clock is used to operate the TRNG. This is for test use only, and indeterminate results may occur.

30.6.3 enum trng_ring_osc_div_t

Used by trng_config_t.

Enumerator

kTRNG_RingOscDiv0 Ring oscillator with no divide.
kTRNG_RingOscDiv2 Ring oscillator divided-by-2.
kTRNG_RingOscDiv4 Ring oscillator divided-by-4.
kTRNG_RingOscDiv8 Ring oscillator divided-by-8.

30.7 Function Documentation

30.7.1 status_t TRNG_GetDefaultConfig (trng_config_t ∗ *userConfig*)

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

```
*     user_config->lock = 0;
*     user_config->clockMode = kTRNG_ClockModeRingOscillator;
*     user_config->ringOscDiv = kTRNG_RingOscDiv0;  Or  to other kTRNG_RingOscDiv[2|8]
      depending on the platform.
*     user_config->sampleMode = kTRNG_SampleModeRaw;
*     user_config->entropyDelay = 3200;
*     user_config->sampleSize = 2500;
*     user_config->sparseBitLimit = TRNG_USER_CONFIG_DEFAULT_SPARSE_BIT_LIMIT;
*     user_config->retryCount = 63;
*     user_config->longRunMaxLimit = 34;
*     user_config->monobitLimit.maximum = 1384;
*     user_config->monobitLimit.minimum = 1116;
*     user_config->runBit1Limit.maximum = 405;
*     user_config->runBit1Limit.minimum = 227;
*     user_config->runBit2Limit.maximum = 220;
*     user_config->runBit2Limit.minimum = 98;
*     user_config->runBit3Limit.maximum = 125;
*     user_config->runBit3Limit.minimum = 37;
*     user_config->runBit4Limit.maximum = 75;
```

```
*     user_config->runBit4Limit.minimum = 11;
*     user_config->runBit5Limit.maximum = 47;
*     user_config->runBit5Limit.minimum = 1;
*     user_config->runBit6PlusLimit.maximum = 47;
*     user_config->runBit6PlusLimit.minimum = 1;
*     user_config->pokerLimit.maximum = 26912;
*     user_config->pokerLimit.minimum = 24445;
*     user_config->frequencyCountLimit.maximum = 25600;
*     user_config->frequencyCountLimit.minimum = 1600;
*
```

Parameters

user_config	User configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

30.7.2 status_t TRNG_Init (TRNG_Type * *base,* const trng_config_t * *userConfig*)

This function initializes the TRNG. When called, the TRNG entropy generation starts immediately.

Parameters

base	TRNG base address
userConfig	Pointer to the initialization configuration structure.

Returns

If successful, returns the kStatus_TRNG_Success. Otherwise, it returns an error.

30.7.3 void TRNG_Deinit (TRNG_Type * *base*)

This function shuts down the TRNG.

Parameters

base	TRNG base address.

30.7.4 status_t TRNG_GetRandomData (TRNG_Type * *base,* void * *data,* size_t *dataSize*)

This function gets random data from the TRNG.

Parameters

base	TRNG base address.
data	Pointer address used to store random data.
dataSize	Size of the buffer pointed by the data parameter.

Returns

random data

Chapter 31
VREF: Voltage Reference Driver

31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Crossbar Voltage Reference (VREF) block of MCUXpresso SDK devices.

The Voltage Reference(VREF) supplies an accurate 1.2 V voltage output that can be trimmed in 0.5 mV steps. VREF can be used in applications to provide a reference voltage to external devices and to internal analog peripherals, such as the ADC, DAC, or CMP. The voltage reference has operating modes that provide different levels of supply rejection and power consumption.

To configure the VREF driver, configure vref_config_t structure in one of two ways.

1. Use the VREF_GetDefaultConfig() function.
2. Set the parameter in the vref_config_t structure.

To initialize the VREF driver, call the VREF_Init() function and pass a pointer to the vref_config_t structure.

To de-initialize the VREF driver, call the VREF_Deinit() function.

31.2 Typical use case and example

This example shows how to generate a reference voltage by using the VREF module.

```
vref_config_t vrefUserConfig;
VREF_GetDefaultConfig(&vrefUserConfig); /* Gets a default configuration. */
VREF_Init(VREF, &vrefUserConfig);       /* Initializes and configures the VREF module */

/* Do something */

VREF_Deinit(VREF); /* De-initializes the VREF module */
```

Data Structures

- struct vref_config_t
  The description structure for the VREF module. More...

Enumerations

- enum vref_buffer_mode_t {
  kVREF_ModeBandgapOnly = 0U,
  kVREF_ModeHighPowerBuffer = 1U,
  kVREF_ModeLowPowerBuffer = 2U }
  VREF modes.

Driver version

- #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))
  Version 2.1.0.

VREF functional operation

- void VREF_Init (VREF_Type *base, const vref_config_t *config)
  Enables the clock gate and configures the VREF module according to the configuration structure.
- void VREF_Deinit (VREF_Type *base)
  Stops and disables the clock for the VREF module.
- void VREF_GetDefaultConfig (vref_config_t *config)
  Initializes the VREF configuration structure.
- void VREF_SetTrimVal (VREF_Type *base, uint8_t trimValue)
  Sets a TRIM value for the reference voltage.
- static uint8_t VREF_GetTrimVal (VREF_Type *base)
  Reads the value of the TRIM meaning output voltage.

31.3 Data Structure Documentation

31.3.1 struct vref_config_t

Data Fields

- vref_buffer_mode_t bufferMode
  Buffer mode selection.

31.4 Macro Definition Documentation

31.4.1 #define FSL_VREF_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))

31.5 Enumeration Type Documentation

31.5.1 enum vref_buffer_mode_t

Enumerator

kVREF_ModeBandgapOnly Bandgap on only, for stabilization and startup.
kVREF_ModeHighPowerBuffer High-power buffer mode enabled.
kVREF_ModeLowPowerBuffer Low-power buffer mode enabled.

31.6 Function Documentation

31.6.1 void VREF_Init (VREF_Type * *base,* const vref_config_t * *config*)

This function must be called before calling all other VREF driver functions, read/write registers, and configurations with user-defined settings. The example below shows how to set up vref_config_t parameters and how to call the VREF_Init function by passing in these parameters. This is an example.

```
*   vref_config_t vrefConfig;
*   vrefConfig.bufferMode = kVREF_ModeHighPowerBuffer;
*   vrefConfig.enableExternalVoltRef = false;
*   vrefConfig.enableLowRef = false;
*   VREF_Init(VREF, &vrefConfig);
*
```

Parameters

base	VREF peripheral address.
config	Pointer to the configuration structure.

31.6.2 void VREF_Deinit (VREF_Type * *base*)

This function should be called to shut down the module. This is an example.

```
*   vref_config_t vrefUserConfig;
*   VREF_Init(VREF);
*   VREF_GetDefaultConfig(&vrefUserConfig);
*   ...
*   VREF_Deinit(VREF);
*
```

Parameters

base	VREF peripheral address.

31.6.3 void VREF_GetDefaultConfig (vref_config_t * *config*)

This function initializes the VREF configuration structure to default values. This is an example.

```
*   vrefConfig->bufferMode = kVREF_ModeHighPowerBuffer;
*   vrefConfig->enableExternalVoltRef = false;
*   vrefConfig->enableLowRef = false;
*
```

Parameters

config	Pointer to the initialization structure.

31.6.4 void VREF_SetTrimVal (VREF_Type * *base,* uint8_t *trimValue*)

This function sets a TRIM value for the reference voltage. Note that the TRIM value maximum is 0x3F.

Parameters

base	VREF peripheral address.
trimValue	Value of the trim register to set the output reference voltage (maximum 0x3F (6-bit)).

31.6.5 static uint8_t VREF_GetTrimVal (VREF_Type * *base*) [inline], [static]

This function gets the TRIM value from the TRM register.

Parameters

base	VREF peripheral address.

Returns

Six-bit value of trim setting.

Chapter 32
Clock Driver

32.1 Overview

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

32.2 Get frequency

A centralized function CLOCK_GetFreq gets different clock type frequencies by passing a clock name. For example, pass a kCLOCK_CoreSysClk to get the core clock and pass a kCLOCK_BusClk to get the bus clock. Additionally, there are separate functions to get the frequency. For example, use CLOCK_GetCoreSysClkFreq to get the core clock frequency and CLOCK_GetBusClkFreq to get the bus clock frequency. Using these functions reduces the image size.

32.3 External clock frequency

The external clocks EXTAL0/EXTAL1/EXTAL32 are decided by the board level design. The Clock driver uses variables g_xtal0Freq/g_xtal1Freq/g_xtal32Freq to save clock frequencies. Likewise, the APIs CLOCK_SetXtal0Freq, CLOCK_SetXtal1Freq, and CLOCK_SetXtal32Freq are used to set these variables.

The upper layer must set these values correctly. For example, after OSC0(SYSOSC) is initialized using CLOCK_InitOsc0 or CLOCK_InitSysOsc, the upper layer should call the CLOCK_SetXtal0Freq. Otherwise, the clock frequency get functions may not receive valid values. This is useful for multicore platforms where only one core calls CLOCK_InitOsc0 to initialize OSC0 and other cores call CLOCK_SetXtal0Freq.

Modules

- Multipurpose Clock Generator (MCG)

Files

- file fsl_clock.h

Data Structures

- struct sim_clock_config_t
  SIM configuration structure for clock setting. More...
- struct osc_config_t
  OSC Initialization Configuration Structure. More...
- struct mcg_config_t
  MCG mode change configuration structure. More...

Macros

- #define MCG_CONFIG_CHECK_PARAM 0U
  Configures whether to check a parameter in a function.
- #define FSL_SDK_DISABLE_DRIVER_CLOCK_CONTROL 0
  Configure whether driver controls clock.
- #define DMAMUX_CLOCKS
  Clock ip name array for DMAMUX.
- #define RTC_CLOCKS
  Clock ip name array for RTC.
- #define PIT_CLOCKS
  Clock ip name array for PIT.
- #define PORT_CLOCKS
  Clock ip name array for PORT.
- #define DSPI_CLOCKS
  Clock ip name array for DSPI.
- #define LPUART_CLOCKS
  Clock ip name array for LPUART.
- #define FLEXCAN_CLOCKS
  Clock ip name array for FLEXCAN.
- #define LPTMR_CLOCKS
  Clock ip name array for LPTMR.
- #define ADC16_CLOCKS
  Clock ip name array for ADC16.
- #define TRNG_CLOCKS
  Clock ip name array for TRNG.
- #define EDMA_CLOCKS
  Clock ip name array for DMA.
- #define CMT_CLOCKS
  Clock ip name array for CMT.
- #define TPM_CLOCKS
  Clock ip name array for TPM.
- #define LTC_CLOCKS
  Clock ip name array for LTC.
- #define I2C_CLOCKS
  Clock ip name array for I2C.
- #define CMP_CLOCKS
  Clock ip name array for CMP.
- #define VREF_CLOCKS
  Clock ip name array for VREF.
- #define DCDC_CLOCKS
  Clock ip name array for DCDC.
- #define LPO_CLK_FREQ 1000U
  LPO clock frequency.
- #define SYS_CLK kCLOCK_CoreSysClk
  Prepherials clock source definition.

Enumerations

- enum clock_name_t {
  kCLOCK_CoreSysClk,
  kCLOCK_PlatClk,
  kCLOCK_BusClk,
  kCLOCK_FlashClk,
  kCLOCK_Er32kClk,
  kCLOCK_Osc0ErClk,
  kCLOCK_McgInternalRefClk,
  kCLOCK_McgFllClk,
  kCLOCK_LpoClk }
  Clock name used to get clock frequency.
- enum clock_ip_name_t
  Clock gate name used for CLOCK_EnableClock/CLOCK_DisableClock.
- enum osc_mode_t {
  kOSC_ModeExt = 0U,
  kOSC_ModeOscLowPower = MCG_C2_EREFS0_MASK,
  kOSC_ModeOscHighGain }
  OSC work mode.
- enum mcg_fll_src_t {
  kMCG_FllSrcExternal,
  kMCG_FllSrcInternal }
  MCG FLL reference clock source select.
- enum mcg_irc_mode_t {
  kMCG_IrcSlow,
  kMCG_IrcFast }
  MCG internal reference clock select.
- enum mcg_dmx32_t {
  kMCG_Dmx32Default,
  kMCG_Dmx32Fine }
  MCG DCO Maximum Frequency with 32.768 kHz Reference.
- enum mcg_drs_t {
  kMCG_DrsLow,
  kMCG_DrsMid,
  kMCG_DrsMidHigh,
  kMCG_DrsHigh }
  MCG DCO range select.
- enum mcg_pll_ref_src_t {
  kMCG_PllRefOsc0,
  kMCG_PllRefOsc1 }
  MCG PLL reference clock select.
- enum mcg_clkout_src_t {
  kMCG_ClkOutSrcOut,
  kMCG_ClkOutSrcInternal,
  kMCG_ClkOutSrcExternal }
  MCGOUT clock source.
- enum mcg_atm_select_t {

kMCG_AtmSel32k,
kMCG_AtmSel4m }
MCG Automatic Trim Machine Select.

- enum mcg_oscsel_t {
kMCG_OscselOsc,
kMCG_OscselRtc }
MCG OSC Clock Select.
- enum mcg_pll_clk_select_t { kMCG_PllClkSelPll0 }
MCG PLLCS select.
- enum mcg_monitor_mode_t {
kMCG_MonitorNone,
kMCG_MonitorInt,
kMCG_MonitorReset }
MCG clock monitor mode.
- enum _mcg_status {
kStatus_MCG_ModeUnreachable = MAKE_STATUS(kStatusGroup_MCG, 0),
kStatus_MCG_ModeInvalid = MAKE_STATUS(kStatusGroup_MCG, 1),
kStatus_MCG_AtmBusClockInvalid = MAKE_STATUS(kStatusGroup_MCG, 2),
kStatus_MCG_AtmDesiredFreqInvalid = MAKE_STATUS(kStatusGroup_MCG, 3),
kStatus_MCG_AtmIrcUsed = MAKE_STATUS(kStatusGroup_MCG, 4),
kStatus_MCG_AtmHardwareFail = MAKE_STATUS(kStatusGroup_MCG, 5),
kStatus_MCG_SourceUsed = MAKE_STATUS(kStatusGroup_MCG, 6) }
MCG status.
- enum _mcg_status_flags_t { kMCG_RtcOscLostFlag = (1U << 4U) }
MCG status flags.
- enum _mcg_irclk_enable_mode {
kMCG_IrclkEnable = MCG_C1_IRCLKEN_MASK,
kMCG_IrclkEnableInStop = MCG_C1_IREFSTEN_MASK }
MCG internal reference clock (MCGIRCLK) enable mode definition.
- enum mcg_mode_t {
kMCG_ModeFEI = 0U,
kMCG_ModeFBI,
kMCG_ModeBLPI,
kMCG_ModeFEE,
kMCG_ModeFBE,
kMCG_ModeBLPE,
kMCG_ModeError }
MCG mode definitions.

Functions

- 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.
- static bool CLOCK_isEnabledClock (clock_ip_name_t name)
Returns true if the clock for specific IP is enabled.
- static void CLOCK_SetEr32kClock (uint32_t src)

Set ERCLK32K source.
- static void CLOCK_SetLpuart0Clock (uint32_t src)
  Set LPUART0 clock source.
- static void CLOCK_SetLpuart1Clock (uint32_t src)
  Set LPUART1 clock source.
- static void CLOCK_SetTpmClock (uint32_t src)
  Set TPM clock source.
- static void CLOCK_SetClkOutClock (uint32_t src)
  Set CLKOUT source.
- static void CLOCK_SetOutDiv (uint32_t outdiv1, uint32_t outdiv4)
  System clock divider.
- uint32_t CLOCK_GetFreq (clock_name_t clockName)
  Gets the clock frequency for a specific clock name.
- uint32_t CLOCK_GetCoreSysClkFreq (void)
  Get the core clock or system clock frequency.
- uint32_t CLOCK_GetPlatClkFreq (void)
  Get the platform clock frequency.
- uint32_t CLOCK_GetBusClkFreq (void)
  Get the bus clock frequency.
- uint32_t CLOCK_GetFlashClkFreq (void)
  Get the flash clock frequency.
- uint32_t CLOCK_GetEr32kClkFreq (void)
  Get the external reference 32K clock frequency (ERCLK32K).
- uint32_t CLOCK_GetOsc0ErClkFreq (void)
  Get the OSC0 external reference clock frequency (OSC0ERCLK).
- void CLOCK_SetSimConfig (sim_clock_config_t const *config)
  Set the clock configure in SIM module.
- static void CLOCK_SetSimSafeDivs (void)
  Set the system clock dividers in SIM to safe value.

Variables

- uint32_t g_xtal0Freq
  External XTAL0 (OSC0) clock frequency.
- uint32_t g_xtal32Freq
  External XTAL32/EXTAL32/RTC_CLKIN clock frequency.

Driver version

- #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))
  CLOCK driver version 2.2.1.

MCG frequency functions.

- uint32_t CLOCK_GetOutClkFreq (void)
  Gets the MCG output clock (MCGOUTCLK) frequency.
- uint32_t CLOCK_GetFllFreq (void)
  Gets the MCG FLL clock (MCGFLLCLK) frequency.
- uint32_t CLOCK_GetInternalRefClkFreq (void)
  Gets the MCG internal reference clock (MCGIRCLK) frequency.
- uint32_t CLOCK_GetFixedFreqClkFreq (void)

Gets the MCG fixed frequency clock (MCGFFCLK) frequency.

MCG clock configuration.

- static void CLOCK_SetLowPowerEnable (bool enable)
  Enables or disables the MCG low power.
- status_t CLOCK_SetInternalRefClkConfig (uint8_t enableMode, mcg_irc_mode_t ircs, uint8_t fcrdiv)
  Configures the Internal Reference clock (MCGIRCLK).
- status_t CLOCK_SetExternalRefClkConfig (mcg_oscsel_t oscsel)
  Selects the MCG external reference clock.
- static void CLOCK_SetFllExtRefDiv (uint8_t frdiv)
  Set the FLL external reference clock divider value.

MCG clock lock monitor functions.

- void CLOCK_SetRtcOscMonitorMode (mcg_monitor_mode_t mode)
  Sets the RTC OSC clock monitor mode.
- uint32_t CLOCK_GetStatusFlags (void)
  Gets the MCG status flags.
- void CLOCK_ClearStatusFlags (uint32_t mask)
  Clears the MCG status flags.

OSC configuration

- void CLOCK_InitOsc0 (osc_config_t const *config)
  Initializes the OSC0.
- void CLOCK_DeinitOsc0 (void)
  Deinitializes the OSC0.

External clock frequency

- static void CLOCK_SetXtal0Freq (uint32_t freq)
  Sets the XTAL0 frequency based on board settings.
- static void CLOCK_SetXtal32Freq (uint32_t freq)
  Sets the XTAL32/RTC_CLKIN frequency based on board settings.

MCG auto-trim machine.

- status_t CLOCK_TrimInternalRefClk (uint32_t extFreq, uint32_t desireFreq, uint32_t *actualFreq, mcg_atm_select_t atms)
  Auto trims the internal reference clock.

MCG mode functions.

- mcg_mode_t CLOCK_GetMode (void)
  Gets the current MCG mode.
- status_t CLOCK_SetFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(*fllStableDelay)(void))
  Sets the MCG to FEI mode.

- status_t CLOCK_SetFeeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FEE mode.
- status_t CLOCK_SetFbiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FBI mode.
- status_t CLOCK_SetFbeMode (uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FBE mode.
- status_t CLOCK_SetBlpiMode (void)
  Sets the MCG to BLPI mode.
- status_t CLOCK_SetBlpeMode (void)
  Sets the MCG to BLPE mode.
- status_t CLOCK_ExternalModeToFbeModeQuick (void)
  Switches the MCG to FBE mode from the external mode.
- status_t CLOCK_InternalModeToFbiModeQuick (void)
  Switches the MCG to FBI mode from internal modes.
- status_t CLOCK_BootToFeiMode (mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FEI mode during system boot up.
- status_t CLOCK_BootToFeeMode (mcg_oscsel_t oscsel, uint8_t frdiv, mcg_dmx32_t dmx32, mcg_drs_t drs, void(∗fllStableDelay)(void))
  Sets the MCG to FEE mode during system bootup.
- status_t CLOCK_BootToBlpiMode (uint8_t fcrdiv, mcg_irc_mode_t ircs, uint8_t ircEnableMode)
  Sets the MCG to BLPI mode during system boot up.
- status_t CLOCK_BootToBlpeMode (mcg_oscsel_t oscsel)
  Sets the MCG to BLPE mode during sytem boot up.
- status_t CLOCK_SetMcgConfig (mcg_config_t const ∗config)
  Sets the MCG to a target mode.

32.4 Data Structure Documentation

32.4.1 struct sim_clock_config_t

Data Fields

- uint8_t pllFllSel
  PLL/FLL/IRC48M selection.
- uint8_t pllFllDiv
  PLLFLLSEL clock divider divisor.
- uint8_t pllFllFrac
  PLLFLLSEL clock divider fraction.
- uint8_t er32kSrc
  ERCLK32K source selection.
- uint32_t clkdiv1
  SIM_CLKDIV1.

32.4.1.0.5.9 Field Documentation

32.4.1.0.5.9.1 uint8_t sim_clock_config_t::pllFllSel

32.4.1.0.5.9.2 uint8_t sim_clock_config_t::pllFllDiv

32.4.1.0.5.9.3 uint8_t sim_clock_config_t::pllFllFrac

32.4.1.0.5.9.4 uint8_t sim_clock_config_t::er32kSrc

32.4.1.0.5.9.5 uint32_t sim_clock_config_t::clkdiv1

32.4.2 struct osc_config_t

Defines the configuration data structure to initialize the OSC. When porting to a new board, set the following members according to the board setting:

1. freq: The external frequency.
2. workMode: The OSC module mode.

Data Fields

- uint32_t freq
  External clock frequency.
- osc_mode_t workMode
  OSC work mode setting.

32.4.2.0.5.10 Field Documentation

32.4.2.0.5.10.1 uint32_t osc_config_t::freq

32.4.2.0.5.10.2 osc_mode_t osc_config_t::workMode

32.4.3 struct mcg_config_t

When porting to a new board, set the following members according to the board setting:

1. frdiv: If the FLL uses the external reference clock, set this value to ensure that the external reference clock divided by frdiv is in the 31.25 kHz to 39.0625 kHz range.
2. The PLL reference clock divider PRDIV: PLL reference clock frequency after PRDIV should be in the FSL_FEATURE_MCG_PLL_REF_MIN to FSL_FEATURE_MCG_PLL_REF_MAX range.

Data Fields

- mcg_mode_t mcgMode
  MCG mode.
- uint8_t irclkEnableMode

MCGIRCLK enable mode.
- mcg_irc_mode_t ircs
  Source, MCG_C2[IRCS].
- uint8_t fcrdiv
  Divider, MCG_SC[FCRDIV].
- uint8_t frdiv
  Divider MCG_C1[FRDIV].
- mcg_drs_t drs
  DCO range MCG_C4[DRST_DRS].
- mcg_dmx32_t dmx32
  MCG_C4[DMX32].
- mcg_oscsel_t oscsel
  OSC select MCG_C7[OSCSEL].

32.4.3.0.5.11 Field Documentation

32.4.3.0.5.11.1 mcg_mode_t mcg_config_t::mcgMode

32.4.3.0.5.11.2 uint8_t mcg_config_t::irclkEnableMode

32.4.3.0.5.11.3 mcg_irc_mode_t mcg_config_t::ircs

32.4.3.0.5.11.4 uint8_t mcg_config_t::fcrdiv

32.4.3.0.5.11.5 uint8_t mcg_config_t::frdiv

32.4.3.0.5.11.6 mcg_drs_t mcg_config_t::drs

32.4.3.0.5.11.7 mcg_dmx32_t mcg_config_t::dmx32

32.4.3.0.5.11.8 mcg_oscsel_t mcg_config_t::oscsel

32.5 Macro Definition Documentation

32.5.1 #define MCG_CONFIG_CHECK_PARAM 0U

Some MCG settings must be changed with conditions, for example:

1. MCGIRCLK settings, such as the source, divider, and the trim value should not change when MCGIRCLK is used as a system clock source.
2. MCG_C7[OSCSEL] should not be changed when the external reference clock is used as a system clock source. For example, in FBE/BLPE/PBE modes.
3. The users should only switch between the supported clock modes.

MCG functions check the parameter and MCG status before setting, if not allowed to change, the functions return error. The parameter checking increases code size, if code size is a critical requirement, change MCG_CONFIG_CHECK_PARAM to 0 to disable parameter checking.

32.5.2 #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 contol 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.

32.5.3 #define FSL_CLOCK_DRIVER_VERSION (MAKE_VERSION(2, 2, 1))

32.5.4 #define DMAMUX_CLOCKS

Value:

```
{                   \
        kCLOCK_Dmamux0 \
    }
```

32.5.5 #define RTC_CLOCKS

Value:

```
{                \
        kCLOCK_Rtc0 \
    }
```

32.5.6 #define PIT_CLOCKS

Value:

```
{                \
        kCLOCK_Pit0 \
    }
```

32.5.7 #define PORT_CLOCKS

Value:

```
{                                          \
        kCLOCK_PortA, kCLOCK_PortB, kCLOCK_PortC \
    }
```

32.5.8 #define DSPI_CLOCKS

Value:

```
{                               \
        kCLOCK_Spi0, kCLOCK_Spi1 \
    }
```

32.5.9 #define LPUART_CLOCKS

Value:

```
{                   \
        kCLOCK_Lpuart0, kCLOCK_Lpuart1 \
    }
```

32.5.10 #define FLEXCAN_CLOCKS

Value:

```
{                   \
        kCLOCK_FlexCan0 \
    }
```

32.5.11 #define LPTMR_CLOCKS

Value:

```
{                  \
        kCLOCK_Lptmr0 \
    }
```

32.5.12 #define ADC16_CLOCKS

Value:

```
{                 \
        kCLOCK_Adc0  \
    }
```

32.5.13 #define TRNG_CLOCKS

Value:

```
{                 \
        kCLOCK_Trng0 \
    }
```

32.5.14 #define EDMA_CLOCKS

Value:

```
{                \
        kCLOCK_Dma0 \
    }
```

32.5.15 #define CMT_CLOCKS

Value:

```
{                \
        kCLOCK_Cmt0 \
    }
```

32.5.16 #define TPM_CLOCKS

Value:

```
{                                          \
        kCLOCK_Tpm0, kCLOCK_Tpm1, kCLOCK_Tpm2 \
    }
```

32.5.17 #define LTC_CLOCKS

Value:

```
{                \
        kCLOCK_Ltc0 \
    }
```

32.5.18 #define I2C_CLOCKS

Value:

```
{                               \
        kCLOCK_I2c0, kCLOCK_I2c1 \
    }
```

32.5.19 #define CMP_CLOCKS

Value:

```
{                \
        kCLOCK_Cmp0 \
    }
```

32.5.20 #define VREF_CLOCKS

Value:

```
{                 \
        kCLOCK_Vref0 \
    }
```

32.5.21 #define DCDC_CLOCKS

Value:

```
{                 \
        kCLOCK_Dcdc0 \
    }
```

32.5.22 #define SYS_CLK kCLOCK_CoreSysClk

32.6 Enumeration Type Documentation

32.6.1 enum clock_name_t

Enumerator

kCLOCK_CoreSysClk Core/system clock.

kCLOCK_PlatClk Platform clock.
kCLOCK_BusClk Bus clock.
kCLOCK_FlashClk Flash clock.
kCLOCK_Er32kClk External reference 32K clock (ERCLK32K)
kCLOCK_Osc0ErClk OSC0 external reference clock (OSC0ERCLK)
kCLOCK_McgInternalRefClk MCG internal reference clock (MCGIRCLK)
kCLOCK_McgFllClk MCGFLLCLK.
kCLOCK_LpoClk LPO clock.

32.6.2 enum clock_ip_name_t

32.6.3 enum osc_mode_t

Enumerator

kOSC_ModeExt Use an external clock.
kOSC_ModeOscLowPower Oscillator low power.
kOSC_ModeOscHighGain Oscillator high gain.

32.6.4 enum mcg_fll_src_t

Enumerator

kMCG_FllSrcExternal External reference clock is selected.
kMCG_FllSrcInternal The slow internal reference clock is selected.

32.6.5 enum mcg_irc_mode_t

Enumerator

kMCG_IrcSlow Slow internal reference clock selected.
kMCG_IrcFast Fast internal reference clock selected.

32.6.6 enum mcg_dmx32_t

Enumerator

kMCG_Dmx32Default DCO has a default range of 25%.
kMCG_Dmx32Fine DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

32.6.7 enum mcg_drs_t

Enumerator

kMCG_DrsLow Low frequency range.
kMCG_DrsMid Mid frequency range.
kMCG_DrsMidHigh Mid-High frequency range.
kMCG_DrsHigh High frequency range.

32.6.8 enum mcg_pll_ref_src_t

Enumerator

kMCG_PllRefOsc0 Selects OSC0 as PLL reference clock.
kMCG_PllRefOsc1 Selects OSC1 as PLL reference clock.

32.6.9 enum mcg_clkout_src_t

Enumerator

kMCG_ClkOutSrcOut Output of the FLL is selected (reset default)
kMCG_ClkOutSrcInternal Internal reference clock is selected.
kMCG_ClkOutSrcExternal External reference clock is selected.

32.6.10 enum mcg_atm_select_t

Enumerator

kMCG_AtmSel32k 32 kHz Internal Reference Clock selected
kMCG_AtmSel4m 4 MHz Internal Reference Clock selected

32.6.11 enum mcg_oscsel_t

Enumerator

kMCG_OscselOsc Selects System Oscillator (OSCCLK)
kMCG_OscselRtc Selects 32 kHz RTC Oscillator.

32.6.12 enum mcg_pll_clk_select_t

Enumerator

kMCG_PllClkSelPll0 PLL0 output clock is selected.

32.6.13 enum mcg_monitor_mode_t

Enumerator

kMCG_MonitorNone Clock monitor is disabled.
kMCG_MonitorInt Trigger interrupt when clock lost.
kMCG_MonitorReset System reset when clock lost.

32.6.14 enum _mcg_status

Enumerator

kStatus_MCG_ModeUnreachable Can't switch to target mode.
kStatus_MCG_ModeInvalid Current mode invalid for the specific function.
kStatus_MCG_AtmBusClockInvalid Invalid bus clock for ATM.
kStatus_MCG_AtmDesiredFreqInvalid Invalid desired frequency for ATM.
kStatus_MCG_AtmIrcUsed IRC is used when using ATM.
kStatus_MCG_AtmHardwareFail Hardware fail occurs during ATM.
kStatus_MCG_SourceUsed Can't change the clock source because it is in use.

32.6.15 enum _mcg_status_flags_t

Enumerator

kMCG_RtcOscLostFlag RTC OSC lost.

32.6.16 enum _mcg_irclk_enable_mode

Enumerator

kMCG_IrclkEnable MCGIRCLK enable.
kMCG_IrclkEnableInStop MCGIRCLK enable in stop mode.

32.6.17 enum mcg_mode_t

Enumerator

kMCG_ModeFEI FEI - FLL Engaged Internal.
kMCG_ModeFBI FBI - FLL Bypassed Internal.
kMCG_ModeBLPI BLPI - Bypassed Low Power Internal.
kMCG_ModeFEE FEE - FLL Engaged External.
kMCG_ModeFBE FBE - FLL Bypassed External.
kMCG_ModeBLPE BLPE - Bypassed Low Power External.
kMCG_ModeError Unknown mode.

32.7 Function Documentation

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

Parameters

name	Which clock to enable, see clock_ip_name_t.

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

Parameters

name	Which clock to disable, see clock_ip_name_t.

32.7.3 static bool CLOCK_isEnabledClock (clock_ip_name_t *name*) [inline], [static]

Parameters

name	Which clock to check for, see clock_ip_name_t.

32.7.4 static void CLOCK_SetEr32kClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set ERCLK32K clock source.

32.7.5 static void CLOCK_SetLpuart0Clock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set LPUART0 clock source.

32.7.6 static void CLOCK_SetLpuart1Clock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set LPUART1 clock source.

32.7.7 static void CLOCK_SetTpmClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set TPM clock source.

32.7.8 static void CLOCK_SetClkOutClock (uint32_t *src*) [inline], [static]

Parameters

src	The value to set CLKOUT source.

32.7.9 static void CLOCK_SetOutDiv (uint32_t *outdiv1,* uint32_t *outdiv4*) [inline], [static]

Set the SIM_CLKDIV1[OUTDIV1], SIM_CLKDIV1[OUTDIV4].

Parameters

outdiv1	Clock 1 output divider value.
outdiv4	Clock 4 output divider value.

32.7.10 uint32_t CLOCK_GetFreq (clock_name_t *clockName*)

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock_name_t. The MCG must be properly configured before using this function.

Parameters

clockName	Clock names defined in clock_name_t

Returns

Clock frequency value in Hertz

32.7.11 uint32_t CLOCK_GetCoreSysClkFreq (void)

Returns

Clock frequency in Hz.

32.7.12 uint32_t CLOCK_GetPlatClkFreq (void)

Returns

Clock frequency in Hz.

32.7.13 uint32_t CLOCK_GetBusClkFreq (void)

Returns

Clock frequency in Hz.

32.7.14 uint32_t CLOCK_GetFlashClkFreq (void)

Returns

Clock frequency in Hz.

32.7.15 uint32_t CLOCK_GetEr32kClkFreq (void)

Returns

Clock frequency in Hz.

32.7.16 uint32_t CLOCK_GetOsc0ErClkFreq (void)

Returns

Clock frequency in Hz.

32.7.17 void CLOCK_SetSimConfig (sim_clock_config_t const * *config*)

This function sets system layer clock settings in SIM module.

Parameters

config	Pointer to the configure structure.

32.7.18 static void CLOCK_SetSimSafeDivs (void) `[inline]`, `[static]`

The system level clocks (core clock, bus clock, flexbus clock and flash clock) must be in allowed ranges. During MCG clock mode switch, the MCG output clock changes then the system level clocks may be out of range. This function could be used before MCG mode change, to make sure system level clocks are in allowed range.

Parameters

config	Pointer to the configure structure.

32.7.19 uint32_t CLOCK_GetOutClkFreq (void)

This function gets the MCG output clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGOUTCLK.

32.7.20 uint32_t CLOCK_GetFllFreq (void)

This function gets the MCG FLL clock frequency in Hz based on the current MCG register value. The FLL is enabled in FEI/FBI/FEE/FBE mode and disabled in low power state in other modes.

Returns

The frequency of MCGFLLCLK.

32.7.21 uint32_t CLOCK_GetInternalRefClkFreq (void)

This function gets the MCG internal reference clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGIRCLK.

32.7.22 uint32_t CLOCK_GetFixedFreqClkFreq (void)

This function gets the MCG fixed frequency clock frequency in Hz based on the current MCG register value.

Returns

The frequency of MCGFFCLK.

32.7.23 static void CLOCK_SetLowPowerEnable (bool *enable*) [inline], [static]

Enabling the MCG low power disables the PLL and FLL in bypass modes. In other words, in FBE and PBE modes, enabling low power sets the MCG to BLPE mode. In FBI and PBI modes, enabling low power sets the MCG to BLPI mode. When disabling the MCG low power, the PLL or FLL are enabled based on MCG settings.

Parameters

enable	True to enable MCG low power, false to disable MCG low power.

32.7.24 status_t CLOCK_SetInternalRefClkConfig (uint8_t *enableMode,* mcg_irc_mode_t *ircs,* uint8_t *fcrdiv*)

This function sets the `MCGIRCLK` base on parameters. It also selects the IRC source. If the fast IRC is used, this function sets the fast IRC divider. This function also sets whether the `MCGIRCLK` is enabled in stop mode. Calling this function in FBI/PBI/BLPI modes may change the system clock. As a result, using the function in these modes it is not allowed.

Parameters

enableMode	MCGIRCLK enable mode, OR'ed value of _mcg_irclk_enable_mode.
ircs	MCGIRCLK clock source, choose fast or slow.
fcrdiv	Fast IRC divider setting (`FCRDIV`).

Return values

kStatus_MCG_Source-Used	Because the internall reference clock is used as a clock source, the confuration should not be changed. Otherwise, a glitch occurs.
kStatus_Success	MCGIRCLK configuration finished successfully.

32.7.25 status_t CLOCK_SetExternalRefClkConfig (mcg_oscsel_t *oscsel*)

Selects the MCG external reference clock source, changes the MCG_C7[OSCSEL], and waits for the clock source to be stable. Because the external reference clock should not be changed in FEE/FBE/BLPE/PBE/PEE modes, do not call this function in these modes.

Parameters

oscsel	MCG external reference clock source, MCG_C7[OSCSEL].

Return values

kStatus_MCG_Source-Used	Because the external reference clock is used as a clock source, the confuration should not be changed. Otherwise, a glitch occurs.
kStatus_Success	External reference clock set successfully.

32.7.26 static void CLOCK_SetFllExtRefDiv (uint8_t *frdiv*) [inline], [static]

Sets the FLL external reference clock divider value, the register MCG_C1[FRDIV].

Parameters

frdiv	The FLL external reference clock divider value, MCG_C1[FRDIV].

32.7.27 void CLOCK_SetRtcOscMonitorMode (mcg_monitor_mode_t *mode*)

This function sets the RTC OSC clock monitor mode. See mcg_monitor_mode_t for details.

Parameters

mode	Monitor mode to set.

32.7.28 uint32_t CLOCK_GetStatusFlags (void)

This function gets the MCG clock status flags. All status flags are returned as a logical OR of the enumeration _mcg_status_flags_t. To check a specific flag, compare the return value with the flag.

Example:

```
// To check the clock lost lock status of OSC0 and PLL0.
uint32_t mcgFlags;

mcgFlags = CLOCK_GetStatusFlags();

if (mcgFlags & kMCG_Osc0LostFlag)
{
    // OSC0 clock lock lost. Do something.
}
if (mcgFlags & kMCG_Pll0LostFlag)
```

```
{
    // PLL0 clock lock lost. Do something.
}
```

Returns

Logical OR value of the _mcg_status_flags_t.

32.7.29 void CLOCK_ClearStatusFlags (uint32_t *mask*)

This function clears the MCG clock lock lost status. The parameter is a logical OR value of the flags to clear. See _mcg_status_flags_t.

Example:

```
// To clear the clock lost lock status flags of OSC0 and PLL0.

CLOCK_ClearStatusFlags(kMCG_Osc0LostFlag | kMCG_Pll0LostFlag);
```

Parameters

mask	The status flags to clear. This is a logical OR of members of the enumeration _mcg_status_flags_t.

32.7.30 void CLOCK_InitOsc0 (osc_config_t const ∗ *config*)

This function initializes the OSC0 according to the board configuration.

Parameters

config	Pointer to the OSC0 configuration structure.

32.7.31 void CLOCK_DeinitOsc0 (void)

This function deinitializes the OSC0.

32.7.32 static void CLOCK_SetXtal0Freq (uint32_t *freq*) [inline], [static]

Parameters

freq	The XTAL0/EXTAL0 input clock frequency in Hz.

32.7.33 static void CLOCK_SetXtal32Freq (uint32_t *freq*) [inline], [static]

Parameters

freq	The XTAL32/EXTAL32/RTC_CLKIN input clock frequency in Hz.

32.7.34 status_t CLOCK_TrimInternalRefClk (uint32_t *extFreq*, uint32_t *desireFreq*, uint32_t * *actualFreq*, mcg_atm_select_t *atms*)

This function trims the internal reference clock by using the external clock. If successful, it returns the kStatus_Success and the frequency after trimming is received in the parameter `actualFreq`. If an error occurs, the error code is returned.

Parameters

extFreq	External clock frequency, which should be a bus clock.
desireFreq	Frequency to trim to.
actualFreq	Actual frequency after trimming.
atms	Trim fast or slow internal reference clock.

Return values

kStatus_Success	ATM success.
kStatus_MCG_AtmBus-ClockInvalid	The bus clock is not in allowed range for the ATM.
kStatus_MCG_Atm-DesiredFreqInvalid	MCGIRCLK could not be trimmed to the desired frequency.
kStatus_MCG_AtmIrc-Used	Could not trim because MCGIRCLK is used as a bus clock source.

kStatus_MCG_Atm-HardwareFail	Hardware fails while trimming.

32.7.35 mcg_mode_t CLOCK_GetMode (void)

This function checks the MCG registers and determines the current MCG mode.

Returns

Current MCG mode or error code; See mcg_mode_t.

32.7.36 status_t CLOCK_SetFeiMode (mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FEI mode. If setting to FEI mode fails from the current mode, this function returns an error.

Parameters

dmx32	DMX32 in FEI mode.
drs	The DCO range selection.
fllStableDelay	Delay function to ensure that the FLL is stable. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

If `dmx32` is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to a frequency above 32768 Hz.

32.7.37 status_t CLOCK_SetFeeMode (uint8_t *frdiv*, mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FEE mode. If setting to FEE mode fails from the current mode, this function returns an error.

Parameters

frdiv	FLL reference clock divider setting, FRDIV.
dmx32	DMX32 in FEE mode.
drs	The DCO range selection.
fllStableDelay	Delay function to make sure FLL is stable. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

32.7.38 status_t CLOCK_SetFbiMode (mcg_dmx32_t *dmx32,* mcg_drs_t *drs,* void(∗)(void) *fllStableDelay*)

This function sets the MCG to FBI mode. If setting to FBI mode fails from the current mode, this function returns an error.

Parameters

dmx32	DMX32 in FBI mode.
drs	The DCO range selection.
fllStableDelay	Delay function to make sure FLL is stable. If the FLL is not used in FBI mode, this parameter can be NULL. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

If `dmx32` is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

32.7.39 status_t CLOCK_SetFbeMode (uint8_t *frdiv,* mcg_dmx32_t *dmx32,* mcg_drs_t *drs,* void(∗)(void) *fllStableDelay*)

This function sets the MCG to FBE mode. If setting to FBE mode fails from the current mode, this function returns an error.

Parameters

frdiv	FLL reference clock divider setting, FRDIV.
dmx32	DMX32 in FBE mode.
drs	The DCO range selection.
fllStableDelay	Delay function to make sure FLL is stable. If the FLL is not used in FBE mode, this parameter can be NULL. Passing NULL does not cause a delay.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

32.7.40 status_t CLOCK_SetBlpiMode (void)

This function sets the MCG to BLPI mode. If setting to BLPI mode fails from the current mode, this function returns an error.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

32.7.41 status_t CLOCK_SetBlpeMode (void)

This function sets the MCG to BLPE mode. If setting to BLPE mode fails from the current mode, this function returns an error.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.

kStatus_Success	Switched to the target mode successfully.

32.7.42 status_t CLOCK_ExternalModeToFbeModeQuick (void)

This function switches the MCG from external modes (PEE/PBE/BLPE/FEE) to the FBE mode quickly. The external clock is used as the system clock souce and PLL is disabled. However, the FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEE mode to FEI mode:

```
* CLOCK_ExternalModeToFbeModeQuick();
* CLOCK_SetFeiMode(...);
*
```

Return values

kStatus_Success	Switched successfully.
kStatus_MCG_Mode-Invalid	If the current mode is not an external mode, do not call this function.

32.7.43 status_t CLOCK_InternalModeToFbiModeQuick (void)

This function switches the MCG from internal modes (PEI/PBI/BLPI/FEI) to the FBI mode quickly. The MCGIRCLK is used as the system clock souce and PLL is disabled. However, FLL settings are not configured. This is a lite function with a small code size, which is useful during the mode switch. For example, to switch from PEI mode to FEE mode:

```
* CLOCK_InternalModeToFbiModeQuick();
* CLOCK_SetFeeMode(...);
*
```

Return values

kStatus_Success	Switched successfully.
kStatus_MCG_Mode-Invalid	If the current mode is not an internal mode, do not call this function.

32.7.44 status_t CLOCK_BootToFeiMode (mcg_dmx32_t *dmx32*, mcg_drs_t *drs*, void(∗)(void) *fllStableDelay*)

This function sets the MCG to FEI mode from the reset mode. It can also be used to set up MCG during system boot up.

Parameters

dmx32	DMX32 in FEI mode.
drs	The DCO range selection.
fllStableDelay	Delay function to ensure that the FLL is stable.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

Note

If `dmx32` is set to kMCG_Dmx32Fine, the slow IRC must not be trimmed to frequency above 32768 Hz.

32.7.45 status_t CLOCK_BootToFeeMode (mcg_oscsel_t *oscsel,* uint8_t *frdiv,* mcg_dmx32_t *dmx32,* mcg_drs_t *drs,* void(∗)(void) *fllStableDelay*)

This function sets MCG to FEE mode from the reset mode. It can also be used to set up the MCG during system boot up.

Parameters

oscsel	OSC clock select, OSCSEL.
frdiv	FLL reference clock divider setting, FRDIV.
dmx32	DMX32 in FEE mode.
drs	The DCO range selection.
fllStableDelay	Delay function to ensure that the FLL is stable.

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.

kStatus_Success	Switched to the target mode successfully.

32.7.46 status_t CLOCK_BootToBlpiMode (uint8_t *fcrdiv,* mcg_irc_mode_t *ircs,* uint8_t *ircEnableMode*)

This function sets the MCG to BLPI mode from the reset mode. It can also be used to set up the MCG during sytem boot up.

Parameters

fcrdiv	Fast IRC divider, FCRDIV.
ircs	The internal reference clock to select, IRCS.
ircEnableMode	The MCGIRCLK enable mode, OR'ed value of _mcg_irclk_enable_mode.

Return values

kStatus_MCG_Source-Used	Could not change MCGIRCLK setting.
kStatus_Success	Switched to the target mode successfully.

32.7.47 status_t CLOCK_BootToBlpeMode (mcg_oscsel_t *oscsel*)

This function sets the MCG to BLPE mode from the reset mode. It can also be used to set up the MCG during sytem boot up.

Parameters

oscsel	OSC clock select, MCG_C7[OSCSEL].

Return values

kStatus_MCG_Mode-Unreachable	Could not switch to the target mode.
kStatus_Success	Switched to the target mode successfully.

32.7.48 status_t CLOCK_SetMcgConfig (mcg_config_t const * *config*)

This function sets MCG to a target mode defined by the configuration structure. If switching to the target mode fails, this function chooses the correct path.

Parameters

config	Pointer to the target MCG mode configuration structure.

Returns

Return kStatus_Success if switched successfully; Otherwise, it returns an error code _mcg_status.

Note

If the external clock is used in the target mode, ensure that it is enabled. For example, if the OSC0 is used, set up OSC0 correctly before calling this function.

32.8 Variable Documentation

32.8.1 uint32_t g_xtal0Freq

The XTAL0/EXTAL0 (OSC0) clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal0Freq to set the value in the clock driver. For example, if XTAL0 is 8 MHz:

```
* CLOCK_InitOsc0(...); // Set up the OSC0
* CLOCK_SetXtal0Freq(80000000); // Set the XTAL0 value to the clock driver.
*
```

This is important for the multicore platforms where only one core needs to set up the OSC0 using the CLOCK_InitOsc0. All other cores need to call the CLOCK_SetXtal0Freq to get a valid clock frequency.

32.8.2 uint32_t g_xtal32Freq

The XTAL32/EXTAL32/RTC_CLKIN clock frequency in Hz. When the clock is set up, use the function CLOCK_SetXtal32Freq to set the value in the clock driver.

This is important for the multicore platforms where only one core needs to set up the clock. All other cores need to call the CLOCK_SetXtal32Freq to get a valid clock frequency.

32.9 Multipurpose Clock Generator (MCG)

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

32.9.1 Function description

MCG driver provides these functions:

- Functions to get the MCG clock frequency.
- Functions to configure the MCG clock, such as PLLCLK and MCGIRCLK.
- Functions for the MCG clock lock lost monitor.
- Functions for the OSC configuration.
- Functions for the MCG auto-trim machine.
- Functions for the MCG mode.

32.9.1.1 MCG frequency functions

MCG module provides clocks, such as MCGOUTCLK, MCGIRCLK, MCGFFCLK, MCGFLLCLK and MCGPLLCLK. The MCG driver provides functions to get the frequency of these clocks, such as CLOCK_GetOutClkFreq(), CLOCK_GetInternalRefClkFreq(), CLOCK_GetFixedFreqClkFreq(), CLOCK_GetFllFreq(), CLOCK_GetPll0Freq(), CLOCK_GetPll1Freq(), and CLOCK_GetExtPllFreq(). These functions get the clock frequency based on the current MCG registers.

32.9.1.2 MCG clock configuration

The MCG driver provides functions to configure the internal reference clock (MCGIRCLK), the external reference clock, and MCGPLLCLK.

The function CLOCK_SetInternalRefClkConfig() configures the MCGIRCLK, including the source and the driver. Do not change MCGIRCLK when the MCG mode is BLPI/FBI/PBI because the MCGIRCLK is used as a system clock in these modes and changing settings makes the system clock unstable.

The function CLOCK_SetExternalRefClkConfig() configures the external reference clock source (MCG_C7[OSCSEL]). Do not call this function when the MCG mode is BLPE/FBE/PBE/FEE/PEE because the external reference clock is used as a clock source in these modes. Changing the external reference clock source requires at least a 50 microseconds wait. The function CLOCK_SetExternalRefClkConfig() implements a for loop delay internally. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 50 micro seconds delay. However, when the system clock is slow, the delay time may significantly increase. This for loop count can be optimized for better performance for specific cases.

The MCGPLLCLK is disabled in FBE/FEE/FBI/FEI modes by default. Applications can enable the MCGPLLCLK in these modes using the functions CLOCK_EnablePll0() and CLOCK_EnablePll1(). To enable the MCGPLLCLK, the PLL reference clock divider(PRDIV) and the PLL VCO divider(VDIV) must be set to a proper value. The function CLOCK_CalcPllDiv() helps to get the PRDIV/VDIV.

32.9.1.3 MCG clock lock monitor functions

The MCG module monitors the OSC and the PLL clock lock status. The MCG driver provides the functions to set the clock monitor mode, check the clock lost status, and clear the clock lost status.

32.9.1.4 OSC configuration

The MCG is needed together with the OSC module to enable the OSC clock. The function CLOCK_InitOsc0() CLOCK_InitOsc1 uses the MCG and OSC to initialize the OSC. The OSC should be configured based on the board design.

32.9.1.5 MCG auto-trim machine

The MCG provides an auto-trim machine to trim the MCG internal reference clock based on the external reference clock (BUS clock). During clock trimming, the MCG must not work in FEI/FBI/BLPI/PBI/PEI modes. The function CLOCK_TrimInternalRefClk() is used for the auto clock trimming.

32.9.1.6 MCG mode functions

The function CLOCK_GetMcgMode returns the current MCG mode. The MCG can only switch between the neighbouring modes. If the target mode is not current mode's neighbouring mode, the application must choose the proper switch path. For example, to switch to PEE mode from FEI mode, use FEI -> FBE -> PBE -> PEE.

For the MCG modes, the MCG driver provides three kinds of functions:

The first type of functions involve functions CLOCK_SetXxxMode, such as CLOCK_SetFeiMode(). These functions only set the MCG mode from neighbouring modes. If switching to the target mode directly from current mode is not possible, the functions return an error.

The second type of functions are the functions CLOCK_BootToXxxMode, such as CLOCK_BootToFeiMode(). These functions set the MCG to specific modes from reset mode. Because the source mode and target mode are specific, these functions choose the best switch path. The functions are also useful to set up the system clock during boot up.

The third type of functions is the CLOCK_SetMcgConfig(). This function chooses the right path to switch to the target mode. It is easy to use, but introduces a large code size.

Whenever the FLL settings change, there should be a 1 millisecond delay to ensure that the FLL is stable. The function CLOCK_SetMcgConfig() implements a for loop delay internally to ensure that the FLL is stable. The for loop delay assumes that the system clock is 96 MHz, which ensures at least 1 millisecond delay. However, when the system clock is slow, the delay time may increase significantly. The for loop count can be optimized for better performance according to a specific use case.

32.9.2 Typical use case

The function CLOCK_SetMcgConfig is used to switch between any modes. However, this heavy-light function introduces a large code size. This section shows how to use the mode function to implement a quick and light-weight switch between typical specific modes. Note that the step to enable the external clock is not included in the following steps. Enable the corresponding clock before using it as a clock source.

32.9.2.1 Switch between BLPI and FEI

Use case	Steps	Functions
BLPI -> FEI	BLPI -> FBI	CLOCK_InternalModeToFbiModeQuick(...)
	FBI -> FEI	CLOCK_SetFeiMode(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
FEI -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
	FEI -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPowerEnable(true)

32.9.2.2 Switch between BLPI and FEE

Use case	Steps	Functions
BLPI -> FEE	BLPI -> FBI	CLOCK_InternalModeToFbiModeQuick(...)
	Change external clock source if need	CLOCK_SetExternalRefClkConfig(...)
	FBI -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClkConfig(...)
	FEE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPowerEnable(true)

32.9.2.3 Switch between BLPI and PEE

Use case	Steps	Functions
BLPI -> PEE	BLPI -> FBI	CLOCK_InternalModeToFbi-ModeQuick(...)
	Change external clock source if need	CLOCK_SetExternalRefClk-Config(...)
	FBI -> FBE	CLOCK_SetFbeMode(...) // fll-StableDelay=NULL
	FBE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPI	PEE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config(...)
	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	FBI -> BLPI	CLOCK_SetLowPower-Enable(true)

32.9.2.4 Switch between BLPE and PEE

This table applies when using the same external clock source (MCG_C7[OSCSEL]) in BLPE mode and PEE mode.

Use case	Steps	Functions
BLPE -> PEE	BLPE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPE	PEE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

If using different external clock sources (MCG_C7[OSCSEL]) in BLPE mode and PEE mode, call the CLOCK_SetExternalRefClkConfig() in FBI or FEI mode to change the external reference clock.

Use case	Steps	Functions
BLPE -> PEE	BLPE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)

	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	FBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> PBE	CLOCK_SetPbeMode(...)
	PBE -> PEE	CLOCK_SetPeeMode(...)
PEE -> BLPE	PEE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)
	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	PBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

32.9.2.5 Switch between BLPE and FEE

This table applies when using the same external clock source (MCG_C7[OSCSEL]) in BLPE mode and FEE mode.

Use case	Steps	Functions
BLPE -> FEE	BLPE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)
	FBE -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPE	PEE -> FBE	CLOCK_SetPbeMode(...)
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

If using different external clock sources (MCG_C7[OSCSEL]) in BLPE mode and FEE mode, call the CLOCK_SetExternalRefClkConfig() in FBI or FEI mode to change the external reference clock.

Use case	Steps	Functions
BLPE -> FEE	BLPE -> FBE	CLOCK_ExternalModeToFbe-ModeQuick(...)

	FBE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	FBI -> FEE	CLOCK_SetFeeMode(...)
FEE -> BLPE	FEE -> FBI	CLOCK_SetFbiMode(...) with fllStableDelay=NULL
	Change source	CLOCK_SetExternalRefClk-Config(...)
	PBI -> FBE	CLOCK_SetFbeMode(...) with fllStableDelay=NULL
	FBE -> BLPE	CLOCK_SetLowPower-Enable(true)

32.9.2.6 Switch between BLPI and PEI

Use case	Steps	Functions
BLPI -> PEI	BLPI -> PBI	CLOCK_SetPbiMode(...)
	PBI -> PEI	CLOCK_SetPeiMode(...)
	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config(...)
PEI -> BLPI	Configure MCGIRCLK if need	CLOCK_SetInternalRefClk-Config
	PEI -> FBI	CLOCK_InternalModeToFbi-ModeQuick(...)
	FBI -> BLPI	CLOCK_SetLowPower-Enable(true)

32.9.3 Code Configuration Option

32.9.3.1 MCG_USER_CONFIG_FLL_STABLE_DELAY_EN

When switching to use FLL with function CLOCK_SetFeiMode() and CLOCK_SetFeeMode(), there is an internal function CLOCK_FllStableDelay(). It is used to delay a few ms so that to wait the FLL to be stable enough. By default, it is implemented in driver code like:

```
#ifndef MCG_USER_CONFIG_FLL_STABLE_DELAY_EN
void CLOCK_FllStableDelay(void)
{
    /*
```

```
       Should wait at least 1ms. Because in these modes, the core clock is 100MHz
       at most, so this function could obtain the 1ms delay.
     */
    volatile uint32_t i = 30000U;
    while (i--)
    {
        __NOP();
    }
}
#endif /* MCG_USER_CONFIG_FLL_STABLE_DELAY_EN */
```

Once user is willing to create his own delay funcion, just assert the macro MCG_USER_CONFIG_FLL-_STABLE_DELAY_EN, and then define function CLOCK_FllStableDelay in the application code.

Chapter 33
DMA Manager

33.1 Overview

DMA Manager provides a series of functions to manage the DMAMUX instances and channels.

33.2 Function groups

33.2.1 DMAMGR Initialization and De-initialization

This function group initializes and deinitializes the DMA Manager.

33.2.2 DMAMGR Operation

This function group requests/releases the DMAMUX channel and configures the channel request source.

33.3 Typical use case

33.3.1 DMAMGR static channel allocattion

```
uint8_t channel;
dmamanager_handle_t dmamanager_handle;

/* Initialize DMAMGR */
DMAMGR_Init(&dmamanager_handle, EXAMPLE_DMA_BASEADDR, DMA_CHANNEL_NUMBER, startChannel);
/* Request a DMAMUX channel by static allocate mechanism */
channel = kDMAMGR_STATIC_ALLOCATE;
DMAMGR_RequestChannel(&dmamanager_handle, kDmaRequestMux0AlwaysOn63, channel, &handle)
      ;
```

33.3.2 DMAMGR dynamic channel allocation

```
uint8_t channel;
dmamanager_handle_t dmamanager_handle;

/* Initialize DMAMGR */
DMAMGR_Init(&dmamanager_handle, EXAMPLE_DMA_BASEADDR, DMA_CHANNEL_NUMBER, startChannel);
/* Request a DMAMUX channel by Dynamic allocate mechanism */
channel = DMAMGR_DYNAMIC_ALLOCATE;
DMAMGR_RequestChannel(&dmamanager_handle, kDmaRequestMux0AlwaysOn63, channel, &handle)
      ;
```

Data Structures

- struct dmamanager_handle_t
  dmamanager handle typedef. *More...*

Macros

- #define DMAMGR_DYNAMIC_ALLOCATE 0xFFU
  Dynamic channel allocation mechanism.

Enumerations

- enum _dma_manager_status {
  kStatus_DMAMGR_ChannelOccupied = MAKE_STATUS(kStatusGroup_DMAMGR, 0),
  kStatus_DMAMGR_ChannelNotUsed = MAKE_STATUS(kStatusGroup_DMAMGR, 1),
  kStatus_DMAMGR_NoFreeChannel = MAKE_STATUS(kStatusGroup_DMAMGR, 2) }
  DMA manager status.

DMAMGR Initialization and De-initialization

- void DMAMGR_Init (dmamanager_handle_t *dmamanager_handle, DMA_Type *dma_base, uint32_t channelNum, uint32_t startChannel)
  Initializes the DMA manager.
- void DMAMGR_Deinit (dmamanager_handle_t *dmamanager_handle)
  Deinitializes the DMA manager.

DMAMGR Operation

- status_t DMAMGR_RequestChannel (dmamanager_handle_t *dmamanager_handle, uint32_t requestSource, uint32_t channel, void *handle)
  Requests a DMA channel.
- status_t DMAMGR_ReleaseChannel (dmamanager_handle_t *dmamanager_handle, void *handle)
  Releases a DMA channel.
- bool DMAMGR_IsChannelOccupied (dmamanager_handle_t *dmamanager_handle, uint32_t channel)
  Get a DMA channel status.

33.4 Data Structure Documentation

33.4.1 struct dmamanager_handle_t

Note

The contents of this structure are private and subject to change.

This dma manager handle structure is used to store the parameters transfered by users.And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.

Data Fields

- void * dma_base
  Peripheral DMA instance.
- uint32_t channelNum

Channel numbers for the DMA instance which need to be managed by dma manager.

- uint32_t startChannel
  The start channel that can be managed by dma manager,users need to transfer it with a certain number or NULL.
- bool s_DMAMGR_Channels [64]
  The s_DMAMGR_Channels is used to store dma manager state.
- uint32_t DmamuxInstanceStart
  The DmamuxInstance is used to calculate the DMAMUX Instance according to the DMA Instance.
- uint32_t multiple
  The multiple is used to calculate the multiple between DMAMUX count and DMA count.

33.4.1.0.0.12 Field Documentation

33.4.1.0.0.12.1 void∗ dmamanager_handle_t::dma_base

33.4.1.0.0.12.2 uint32_t dmamanager_handle_t::channelNum

33.4.1.0.0.12.3 uint32_t dmamanager_handle_t::startChannel

33.4.1.0.0.12.4 bool dmamanager_handle_t::s_DMAMGR_Channels[64]

33.4.1.0.0.12.5 uint32_t dmamanager_handle_t::DmamuxInstanceStart

33.4.1.0.0.12.6 uint32_t dmamanager_handle_t::multiple

33.5 Macro Definition Documentation

33.5.1 #define DMAMGR_DYNAMIC_ALLOCATE 0xFFU

33.6 Enumeration Type Documentation

33.6.1 enum _dma_manager_status

Enumerator

kStatus_DMAMGR_ChannelOccupied Channel has been occupied.
kStatus_DMAMGR_ChannelNotUsed Channel has not been used.
kStatus_DMAMGR_NoFreeChannel All channels have been occupied.

33.7 Function Documentation

33.7.1 void DMAMGR_Init (dmamanager_handle_t ∗ *dmamanager_handle,* DMA_Type ∗ *dma_base,* uint32_t *channelNum,* uint32_t *startChannel*)

This function initializes the DMA manager, ungates the DMAMUX clocks, and initializes the eDMA or DMA peripherals.

Parameters

dmamanager_-handle	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
dma_base	Peripheral DMA instance base pointer.
dmamux_base	Peripheral DMAMUX instance base pointer.
channelNum	Channel numbers for the DMA instance which need to be managed by dma manager.
startChannel	The start channel that can be managed by dma manager.

33.7.2 void DMAMGR_Deinit (dmamanager_handle_t ∗ *dmamanager_handle*)

This function deinitializes the DMA manager, disables the DMAMUX channels, gates the DMAMUX clocks, and deinitializes the eDMA or DMA peripherals.

Parameters

dmamanager_-handle	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.

33.7.3 status_t DMAMGR_RequestChannel (dmamanager_handle_t ∗ *dmamanager_handle,* uint32_t *requestSource,* uint32_t *channel,* void ∗ *handle*)

This function requests a DMA channel which is not occupied. The two channels to allocate the mechanism are dynamic and static channels. For the dynamic allocation mechanism (channe = DMAMGR_DYNAMIC_ALLOCATE), DMAMGR allocates a DMA channel according to the given request source and startChannel and then configures it. For static allocation mechanism, DMAMGR configures the given channel according to the given request source and channel number.

Parameters

dmamanager_-handle	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
requestSource	DMA channel request source number. See the soc.h, see the enum dma_request_-source_t
channel	The channel number users want to occupy. If using the dynamic channel allocate mechanism, set the channel equal to DMAMGR_DYNAMIC_ALLOCATE.
handle	DMA or eDMA handle pointer.

Return values

kStatus_Success	In a dynamic/static channel allocation mechanism, allocate the DMAMUX channel successfully.
kStatus_DMAMGR_No-FreeChannel	In a dynamic channel allocation mechanism, all DMAMUX channels are occupied.
kStatus_DMAMGR_-ChannelOccupied	In a static channel allocation mechanism, the given channel is occupied.

33.7.4 status_t DMAMGR_ReleaseChannel (dmamanager_handle_t * *dmamanager_handle,* void * *handle*)

This function releases an occupied DMA channel.

Parameters

dmamanager_-handle	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
handle	DMA or eDMA handle pointer.

Return values

kStatus_Success	Releases the given channel successfully.
kStatus_DMAMGR_-ChannelNotUsed	The given channel to be released had not been used before.

33.7.5 bool DMAMGR_IsChannelOccupied (dmamanager_handle_t * *dmamanager_handle,* uint32_t *channel*)

This function get a DMA channel status. Return 0 indicates the channel has not been used, return 1 indicates the channel has been occupied.

Parameters

dmamanager_-handle	DMA manager handle pointer, this structure is maintained by dma manager internal,users only need to transfer the structure to the function. And users shall not free the memory before calling DMAMGR_Deinit, also shall not modify the contents of the memory.
channel	The channel number that users want get its status.

Chapter 34
Secure Digital Card/Embedded MultiMedia Card (CARD)

34.1 Overview

The MCUXpresso SDK provides a driver to access the Secure Digital Card and Embedded MultiMedia Card based on the SDHC driver.

Function groups

This function group implements the SD card functional API.

This function group implements the MMC card functional API.

Typical use case

```
/* Initialize SDHC. */
sdhcConfig->cardDetectDat3 = false;
sdhcConfig->endianMode = kSDHC_EndianModeLittle;
sdhcConfig->dmaMode = kSDHC_DmaModeAdma2;
sdhcConfig->readWatermarkLevel = 0x80U;
sdhcConfig->writeWatermarkLevel = 0x80U;
SDHC_Init(BOARD_SDHC_BASEADDR, sdhcConfig);

/* Save host information. */
card->host.base = BOARD_SDHC_BASEADDR;
card->host.sourceClock_Hz = CLOCK_GetFreq(BOARD_SDHC_CLKSRC);
card->host.transfer = SDHC_TransferFunction;

/* Init card. */
if (SD_Init(card))
{
    PRINTF("\r\nSD card init failed.\r\n");
}

while (true)
{
    if (kStatus_Success != SD_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START,
      DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != SD_ReadBlocks(card, g_dataRead, DATA_BLOCK_START, DATA_BLOCK_COUNT)
      )
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }

    if (kStatus_Success != SD_EraseBlocks(card, DATA_BLOCK_START, DATA_BLOCK_COUNT))
    {
        PRINTF("Erase multiple data blocks failed.\r\n");
    }
}

SD_Deinit(card);


/* Initialize SDHC. */
```

```
sdhcConfig->cardDetectDat3 = false;
sdhcConfig->endianMode = kSDHC_EndianModeLittle;
sdhcConfig->dmaMode = kSDHC_DmaModeAdma2;
sdhcConfig->readWatermarkLevel = 0x80U;
sdhcConfig->writeWatermarkLevel = 0x80U;
SDHC_Init(BOARD_SDHC_BASEADDR, sdhcConfig);

/* Save host information. */
card->host.base = BOARD_SDHC_BASEADDR;
card->host.sourceClock_Hz = CLOCK_GetFreq(BOARD_SDHC_CLKSRC);
card->host.transfer = SDHC_TransferFunction;

/* Init card. */
if (MMC_Init(card))
{
    PRINTF("\n MMC card init failed \n");
}

while (true)
{
    if (kStatus_Success != MMC_WriteBlocks(card, g_dataWrite, DATA_BLOCK_START,
      DATA_BLOCK_COUNT))
    {
        PRINTF("Write multiple data blocks failed.\r\n");
    }
    if (kStatus_Success != MMC_ReadBlocks(card, g_dataRead, DATA_BLOCK_START,
      DATA_BLOCK_COUNT))
    {
        PRINTF("Read multiple data blocks failed.\r\n");
    }
}

MMC_Deinit(card);
```

Data Structures

- struct sd_card_t
  SD card state. More...
- struct sdio_card_t
  SDIO card state. More...
- struct mmc_card_t
  SD card state. More...
- struct mmc_boot_config_t
  MMC card boot configuration definition. More...

Macros

- #define FSL_SDMMC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 4U)) /∗2.1.4∗/
  Driver version.
- #define FSL_SDMMC_DEFAULT_BLOCK_SIZE (512U)
  Default block size.
- #define SDMMC_GLOBAL_BUFFER_SIZE (64U)
  SDMMC global data buffer size, word unit.
- #define HOST_NOT_SUPPORT 0U
  use this define to indicate the host not support feature
- #define HOST_SUPPORT 1U
  use this define to indicate the host support feature

Enumerations

- enum _sdmmc_status {
  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_ReTuningRequest = MAKE_STATUS(kStatusGroup_SDMMC, 35U),
  kStatus_SDMMC_SetDriverStrengthFail = MAKE_STATUS(kStatusGroup_SDMMC, 36U),
  kStatus_SDMMC_SetPowerClassFail = MAKE_STATUS(kStatusGroup_SDMMC, 37U) }
  SD/MMC card API's running status.
- enum _sd_card_flag {

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.

- enum _mmc_card_flag {
  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_DataBusWidth4BitFlag = (1U << 12U),
  kMMC_DataBusWidth8BitFlag = (1U << 13U),
  kMMC_DataBusWidth1BitFlag = (1U << 14U) }

  MMC card flags.
- enum card_operation_voltage_t {
  kCARD_OperationVoltageNone = 0U,
  kCARD_OperationVoltage330V = 1U,
  kCARD_OperationVoltage300V = 2U,
  kCARD_OperationVoltage180V = 3U }

  card operation voltage
- enum _host_endian_mode {
  kHOST_EndianModeBig = 0U,
  kHOST_EndianModeHalfWordBig = 1U,
  kHOST_EndianModeLittle = 2U }

  host Endian mode corresponding to driver define

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.
- bool SD_CheckReadOnly (sd_card_t *card)

  Checks whether the card is write-protected.
- status_t SD_ReadBlocks (sd_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)

Reads blocks from the specific card.
- status_t SD_WriteBlocks (sd_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)
  Writes blocks of data to the specific card.
- status_t SD_EraseBlocks (sd_card_t *card, uint32_t startBlock, uint32_t blockCount)
  Erases blocks of the specific card.

MMCCARD Function

- status_t MMC_Init (mmc_card_t *card)
  Initializes the MMC card.
- void MMC_Deinit (mmc_card_t *card)
  Deinitializes the card.
- 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 SDIO_CardInActive (sdio_card_t *card)
  set SDIO card to inactive state
- 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_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_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_Init (sdio_card_t *card)
  SDIO card init 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.
- void SDIO_DeInit (sdio_card_t *card)
  SDIO card deinit.

adaptor function

- static status_t HOST_NotSupport (void *parameter)
  host not support function, this function is used for host not support feature
- status_t CardInsertDetect (HOST_TYPE *hostBase)
  Detect card insert, only need for SD cases.
- status_t HOST_Init (void *host)
  Init host controller.
- void HOST_Reset (HOST_TYPE *hostBase)
  reset host controller.
- void HOST_Deinit (void *host)
  Deinit host controller.

34.2 Data Structure Documentation

34.2.1 struct sd_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- HOST_CONFIG host
  Host information.
- bool isHostReady
  use this flag to indicate if need host re-init or not
- 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.
- uint32_t rawCid [4U]
  Raw CID content.
- uint32_t rawCsd [4U]
  Raw CSD content.
- uint32_t rawScr [2U]
  Raw CSD content.
- uint32_t ocr
  Raw OCR content.
- sd_cid_t cid
  CID.
- sd_csd_t csd
  CSD.
- sd_scr_t scr
  SCR.
- 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
- card_operation_voltage_t operationVoltage
  card operation voltage

34.2.2 struct sdio_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- HOST_CONFIG host
  Host information.
- 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.
- 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 avalible for SDIO card.
- uint32_t commonCISPointer
  point to common CIS
- sdio_fbr_t ioFBR [7U]
  FBR table.
- sdio_common_cis_t commonCIS
  CIS table.
- sdio_func_cis_t funcCIS [7U]
  function CIS table

34.2.3 struct mmc_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- HOST_CONFIG host
  Host information.
- bool isHostReady
  use this flag to indicate if need host re-init or not
- 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.
- uint32_t rawCid [4U]
  Raw CID content.
- uint32_t rawCsd [4U]
  Raw CSD content.
- uint32_t rawExtendedCsd [MMC_EXTENDED_CSD_BYTES/4U]
  Raw MMC Extended CSD content.
- 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
  Host IO voltage window.
- mmc_voltage_window_t hostVoltageWindowVCC
  application must set this value according to board specific
- mmc_high_speed_timing_t currentTiming
  indicate the current host timing mode

34.2.4 struct mmc_boot_config_t

Data Fields

- bool enableBootAck
  Enable boot ACK.
- mmc_boot_partition_enable_t bootPartition
  Boot partition.
- bool retainBootBusWidth
  If retain boot bus width.
- mmc_data_bus_width_t bootDataBusWidth
  Boot data bus width.

34.3 Macro Definition Documentation

34.3.1 #define FSL_SDMMC_DRIVER_VERSION (MAKE_VERSION(2U, 1U, 4U)) /∗2.1.4∗/

34.4 Enumeration Type Documentation

34.4.1 enum _sdmmc_status

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 invaild voltage
kStatus_SDMMC_SDIO_SwitchHighSpeedFail switch to high speed fail
kStatus_SDMMC_SDIO_ReadCISFail read CIS fail
kStatus_SDMMC_SDIO_InvalidCard invaild SDIO card
kStatus_SDMMC_TuningFail tuning fail
kStatus_SDMMC_SwitchVoltageFail switch voltage fail
kStatus_SDMMC_ReTuningRequest retuning request
kStatus_SDMMC_SetDriverStrengthFail set driver strength fail
kStatus_SDMMC_SetPowerClassFail set power class fail

34.4.2 enum _sd_card_flag

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

34.4.3 enum _mmc_card_flag

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_DataBusWidth4BitFlag current data bus is 4 bit mode
kMMC_DataBusWidth8BitFlag current data bus is 8 bit mode
kMMC_DataBusWidth1BitFlag current data bus is 1 bit mode

34.4.4 enum card_operation_voltage_t

Enumerator

kCARD_OperationVoltageNone indicate current voltage setting is not setting bu suser
kCARD_OperationVoltage330V card operation voltage around 3.3v
kCARD_OperationVoltage300V card operation voltage around 3.0v
kCARD_OperationVoltage180V card operation voltage around 31.8v

34.4.5 enum _host_endian_mode

Enumerator

kHOST_EndianModeBig Big endian mode.
kHOST_EndianModeHalfWordBig Half word big endian mode.
kHOST_EndianModeLittle Little endian mode.

34.5 Function Documentation

34.5.1 status_t SD_Init (sd_card_t * *card*)

This function initializes the card on a specific host controller.

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_NotSupportYet	Card not support.
kStatus_SDMMC_SendOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SendRelativeAddressFailed	Send relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendScrFailed	Send SCR failed.
kStatus_SDMMC_SetBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchHighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

34.5.2 void SD_Deinit (sd_card_t ∗ *card*)

This function deinitializes the specific card.

Parameters

card	Card descriptor.

34.5.3 bool SD_CheckReadOnly (sd_card_t * *card*)

This function checks if the card is write-protected via the CSD register.

Parameters

card	The specific card.

Return values

true	Card is read only.
false	Card isn't read only.

34.5.4 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.

Parameters

card	Card descriptor.
buffer	The buffer to save the data read from card.
startBlock	The start block index.
blockCount	The number of blocks to read.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_Card-NotSupport	Card not support.

kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

34.5.5 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.

Parameters

card	Card descriptor.
buffer	The buffer holding the data to be written to the card.
startBlock	The start block index.
blockCount	The number of blocks to write.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.

kStatus_SDMMC_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

34.5.6 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.

Parameters

card	Card descriptor.
startBlock	The start block index.
blockCount	The number of blocks to erase.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_Success	Operate successfully.

34.5.7 status_t MMC_Init (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_GoIdleFailed	Go idle failed.
kStatus_SDMMC_SendOperationConditionFailed	Send operation condition failed.
kStatus_SDMMC_AllSendCidFailed	Send CID failed.
kStatus_SDMMC_SetRelativeAddressFailed	Set relative address failed.
kStatus_SDMMC_SendCsdFailed	Send CSD failed.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_SelectCardFailed	Send SELECT_CARD command failed.
kStatus_SDMMC_SendExtendedCsdFailed	Send EXT_CSD failed.
kStatus_SDMMC_SetBusWidthFailed	Set bus width failed.
kStatus_SDMMC_SwitchHighSpeedFailed	Switch high speed failed.
kStatus_SDMMC_SetCardBlockSizeFailed	Set card block size failed.
kStatus_Success	Operate successfully.

34.5.8 void MMC_Deinit (mmc_card_t * *card*)

Parameters

card	Card descriptor.

34.5.9 bool MMC_CheckReadOnly (mmc_card_t * *card*)

Parameters

card	Card descriptor.

Return values

true	Card is read only.
false	Card isn't read only.

34.5.10 status_t MMC_ReadBlocks (mmc_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Parameters

card	Card descriptor.
buffer	The buffer to save data.
startBlock	The start block index.
blockCount	The number of blocks to read.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_CardNotSupport	Card not support.
kStatus_SDMMC_SetBlockCountFailed	Set block count failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

34.5.11 status_t MMC_WriteBlocks (mmc_card_t * *card*, const uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

Parameters

card	Card descriptor.
buffer	The buffer to save data blocks.
startBlock	Start block number to write.
blockCount	Block count.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_SetBlockCountFailed	Set block count failed.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_SDMMC_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

34.5.12 status_t MMC_EraseGroups (mmc_card_t * *card*, uint32_t *startGroup*, uint32_t *endGroup*)

Erase group is the smallest erase unit in MMC card. The erase range is [startGroup, endGroup].

Parameters

card	Card descriptor.
startGroup	Start group number.
endGroup	End group number.

Return values

kStatus_InvalidArgument	Invalid argument.
kStatus_SDMMC_WaitWriteCompleteFailed	Send status failed.
kStatus_SDMMC_TransferFailed	Transfer failed.
kStatus_Success	Operate successfully.

34.5.13 status_t MMC_SelectPartition (mmc_card_t * *card,* mmc_access_partition_t *partitionNumber*)

Parameters

card	Card descriptor.
partitionNumber	The partition number.

Return values

kStatus_SDMMC_ConfigureExtendedCsdFailed	Configure EXT_CSD failed.
kStatus_Success	Operate successfully.

34.5.14 status_t MMC_SetBootConfig (mmc_card_t * *card,* const mmc_boot_config_t * *config*)

Parameters

card	Card descriptor.
config	Boot configuration structure.

Return values

kStatus_SDMMC_NotSupportYet	Not support now.
kStatus_SDMMC_ConfigureExtendedCsdFailed	Configure EXT_CSD failed.
kStatus_SDMMC_ConfigureBootFailed	Configure boot failed.
kStatus_Success	Operate successfully.

34.5.15 status_t SDIO_CardInActive (sdio_card_t ∗ *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_TransferFailed	
kStatus_Success	

34.5.16 status_t SDIO_IO_Write_Direct (sdio_card_t ∗ *card,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t ∗ *data,* bool *raw*)

Parameters

card	Card descriptor.
function	IO numner
register	address
the	data pinter to write
raw	flag, indicate read after write or write only

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.17 status_t SDIO_IO_Read_Direct (sdio_card_t * *card,* sdio_func_num_t *func,* uint32_t *regAddr,* uint8_t * *data*)

Parameters

card	Card descriptor.
function	IO number
register	address
data	pointer to read

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.18 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*)

Parameters

card	Card descriptor.
function	IO number
register	address
data	buffer to write
data	count
write	flags

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

34.5.19 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*)

Parameters

card	Card descriptor.
function	IO number
register	address
data	buffer to read
data	count
write	flags

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

34.5.20 status_t SDIO_GetCardCapability (sdio_card_t * *card,* sdio_func_num_t *func*)

Parameters

card	Card descriptor.
function	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.21 status_t SDIO_SetBlockSize (sdio_card_t * *card,* sdio_func_num_t *func,* uint32_t *blockSize*)

Parameters

card	Card descriptor.
function	io number
block	size

Return values

kStatus_SDMMC_Set-CardBlockSizeFailed	
kStatus_SDMMC_SDIO-_InvalidArgument	
kStatus_Success	

34.5.22 status_t SDIO_CardReset (sdio_card_t * *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.23 status_t SDIO_SetDataBusWidth (sdio_card_t * *card,* sdio_bus_width_t *busWidth*)

Parameters

card	Card descriptor.
data	bus width

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.24 status_t SDIO_SwitchToHighSpeed (sdio_card_t ∗ *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_SDMMC_SDIO-_SwitchHighSpeedFail	
kStatus_Success	

34.5.25 status_t SDIO_ReadCIS (sdio_card_t ∗ *card,* sdio_func_num_t *func,* const uint32_t ∗ *tupleList,* uint32_t *tupleNum*)

Parameters

card	Card descriptor.
function	io number
tuple	code list
tuple	code number

Return values

kStatus_SDMMC_SDIO-_ReadCISFail	
kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.26 status_t SDIO_Init (sdio_card_t ∗ *card*)

Parameters

card	Card descriptor.

Return values

kStatus_SDMMC_Go-IdleFailed	
kStatus_SDMMC_Hand-ShakeOperation-ConditionFailed	
kStatus_SDMMC_SDIO-_InvalidCard	
kStatus_SDMMC_SDIO-_InvalidVoltage	
kStatus_SDMMC_Send-RelativeAddressFailed	
kStatus_SDMMC_Select-CardFailed	
kStatus_SDMMC_SDIO-_SwitchHighSpeedFail	
kStatus_SDMMC_SDIO-_ReadCISFail	
kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.27 status_t SDIO_EnableIOInterrupt (sdio_card_t * *card,* sdio_func_num_t *func,* bool *enable*)

Parameters

card	Card descriptor.
function	IO number
enable/disable	flag

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.28 status_t SDIO_EnableIO (sdio_card_t * *card,* sdio_func_num_t *func,* bool *enable*)

Parameters

card	Card descriptor.
function	IO number
enable/disable	flag

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.29 status_t SDIO_SelectIO (sdio_card_t * *card,* sdio_func_num_t *func*)

Parameters

card	Card descriptor.
function	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.30 status_t SDIO_AbortIO (sdio_card_t * *card,* sdio_func_num_t *func*)

Parameters

card	Card descriptor.
function	IO number

Return values

kStatus_SDMMC_-TransferFailed	
kStatus_Success	

34.5.31 void SDIO_DeInit (sdio_card_t * *card*)

Parameters

card	Card descriptor.

34.5.32 static status_t HOST_NotSupport (void * *parameter*) [inline], [static]

Parameters

void	parameter ,used to avoid build warning

Return values

kStatus_Fail,host	do not suppport

34.5.33 status_t CardInsertDetect (HOST_TYPE * *hostBase*)

Parameters

hostBase	the pointer to host base address

Return values

kStatus_Success	detect card insert
kStatus_Fail	card insert event fail

34.5.34 status_t HOST_Init (void ∗ *host*)

Parameters

host	the pointer to host structure in card structure.

Return values

kStatus_Success	host init success
kStatus_Fail	event fail

34.5.35 void HOST_Reset (HOST_TYPE ∗ *hostBase*)

Parameters

host	base address.

34.5.36 void HOST_Deinit (void ∗ *host*)

Parameters

host	the pointer to host structure in card structure.

Chapter 35
SPI based Secure Digital Card (SDSPI)

35.1 Overview

The MCUXpresso SDK provides a driver to access the Secure Digital Card based on the SPI driver.

Function groups

This function group implements the SD card functional API in the SPI mode.

Typical use case

```
/* SPI_Init(). */

/* Register the SDSPI driver callback. */

/* Initializes card. */
if (kStatus_Success != SDSPI_Init(card))
{
    SDSPI_Deinit(card)
    return;
}

/* Read/Write card */
memset(g_testWriteBuffer, 0x17U, sizeof(g_testWriteBuffer));

while (true)
{
    memset(g_testReadBuffer, 0U, sizeof(g_testReadBuffer));

    SDSPI_WriteBlocks(card, g_testWriteBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    SDSPI_ReadBlocks(card, g_testReadBuffer, TEST_START_BLOCK, TEST_BLOCK_COUNT);

    if (memcmp(g_testReadBuffer, g_testReadBuffer, sizeof(g_testWriteBuffer)))
    {
        break;
    }
}
```

Data Structures

- struct sdspi_command_t
  SDSPI command. More...
- struct sdspi_host_t
  SDSPI host state. More...
- struct sdspi_card_t
  SD Card Structure. More...

Enumerations

- enum _sdspi_status {
  kStatus_SDSPI_SetFrequencyFailed = MAKE_STATUS(kStatusGroup_SDSPI, 0U),
  kStatus_SDSPI_ExchangeFailed = MAKE_STATUS(kStatusGroup_SDSPI, 1U),
  kStatus_SDSPI_WaitReadyFailed = MAKE_STATUS(kStatusGroup_SDSPI, 2U),
  kStatus_SDSPI_ResponseError = MAKE_STATUS(kStatusGroup_SDSPI, 3U),
  kStatus_SDSPI_WriteProtected = MAKE_STATUS(kStatusGroup_SDSPI, 4U),
  kStatus_SDSPI_GoIdleFailed = MAKE_STATUS(kStatusGroup_SDSPI, 5U),
  kStatus_SDSPI_SendCommandFailed = MAKE_STATUS(kStatusGroup_SDSPI, 6U),
  kStatus_SDSPI_ReadFailed = MAKE_STATUS(kStatusGroup_SDSPI, 7U),
  kStatus_SDSPI_WriteFailed = MAKE_STATUS(kStatusGroup_SDSPI, 8U),
  kStatus_SDSPI_SendInterfaceConditionFailed,
  kStatus_SDSPI_SendOperationConditionFailed,
  kStatus_SDSPI_ReadOcrFailed = MAKE_STATUS(kStatusGroup_SDSPI, 11U),
  kStatus_SDSPI_SetBlockSizeFailed = MAKE_STATUS(kStatusGroup_SDSPI, 12U),
  kStatus_SDSPI_SendCsdFailed = MAKE_STATUS(kStatusGroup_SDSPI, 13U),
  kStatus_SDSPI_SendCidFailed = MAKE_STATUS(kStatusGroup_SDSPI, 14U),
  kStatus_SDSPI_StopTransmissionFailed = MAKE_STATUS(kStatusGroup_SDSPI, 15U),
  kStatus_SDSPI_SendApplicationCommandFailed }
  SDSPI API status.
- enum _sdspi_card_flag {
  kSDSPI_SupportHighCapacityFlag = (1U << 0U),
  kSDSPI_SupportSdhcFlag = (1U << 1U),
  kSDSPI_SupportSdxcFlag = (1U << 2U),
  kSDSPI_SupportSdscFlag = (1U << 3U) }
  SDSPI card flag.
- enum sdspi_response_type_t {
  kSDSPI_ResponseTypeR1 = 0U,
  kSDSPI_ResponseTypeR1b = 1U,
  kSDSPI_ResponseTypeR2 = 2U,
  kSDSPI_ResponseTypeR3 = 3U,
  kSDSPI_ResponseTypeR7 = 4U }
  SDSPI response type.

SDSPI Function

- status_t SDSPI_Init (sdspi_card_t *card)
  Initializes the card on a specific SPI instance.
- void SDSPI_Deinit (sdspi_card_t *card)
  Deinitializes the card.
- bool SDSPI_CheckReadOnly (sdspi_card_t *card)
  Checks whether the card is write-protected.
- status_t SDSPI_ReadBlocks (sdspi_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)
  Reads blocks from the specific card.
- status_t SDSPI_WriteBlocks (sdspi_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount)

Writes blocks of data to the specific card.

35.2 Data Structure Documentation

35.2.1 struct sdspi_command_t

Data Fields

- uint8_t index
  Command index.
- uint32_t argument
  Command argument.
- uint8_t responseType
  Response type.
- uint8_t response [5U]
  Response content.

35.2.2 struct sdspi_host_t

Data Fields

- uint32_t busBaudRate
  Bus baud rate.
- status_t(∗ setFrequency)(uint32_t frequency)
  Set frequency of SPI.
- status_t(∗ exchange)(uint8_t ∗in, uint8_t ∗out, uint32_t size)
  Exchange data over SPI.
- uint32_t(∗ getCurrentMilliseconds)(void)
  Get current time in milliseconds.

35.2.3 struct sdspi_card_t

Define the card structure including the necessary fields to identify and describe the card.

Data Fields

- sdspi_host_t ∗ host
  Host state information.
- uint32_t relativeAddress
  Relative address of the card.
- uint32_t flags
  Flags defined in _sdspi_card_flag.
- uint8_t rawCid [16U]
  Raw CID content.
- uint8_t rawCsd [16U]

Raw CSD content.
- uint8_t rawScr [8U]
  Raw SCR content.
- uint32_t ocr
  Raw OCR content.
- sd_cid_t cid
  CID.
- sd_csd_t csd
  CSD.
- sd_scr_t scr
  SCR.
- uint32_t blockCount
  Card total block number.
- uint32_t blockSize
  Card block size.

35.2.3.0.0.13 Field Documentation

35.2.3.0.0.13.1 uint32_t sdspi_card_t::flags

35.3 Enumeration Type Documentation

35.3.1 enum _sdspi_status

Enumerator

kStatus_SDSPI_SetFrequencyFailed Set frequency failed.
kStatus_SDSPI_ExchangeFailed Exchange data on SPI bus failed.
kStatus_SDSPI_WaitReadyFailed Wait card ready failed.
kStatus_SDSPI_ResponseError Response is error.
kStatus_SDSPI_WriteProtected Write protected.
kStatus_SDSPI_GoIdleFailed Go idle failed.
kStatus_SDSPI_SendCommandFailed Send command failed.
kStatus_SDSPI_ReadFailed Read data failed.
kStatus_SDSPI_WriteFailed Write data failed.
kStatus_SDSPI_SendInterfaceConditionFailed Send interface condition failed.
kStatus_SDSPI_SendOperationConditionFailed Send operation condition failed.
kStatus_SDSPI_ReadOcrFailed Read OCR failed.
kStatus_SDSPI_SetBlockSizeFailed Set block size failed.
kStatus_SDSPI_SendCsdFailed Send CSD failed.
kStatus_SDSPI_SendCidFailed Send CID failed.
kStatus_SDSPI_StopTransmissionFailed Stop transmission failed.
kStatus_SDSPI_SendApplicationCommandFailed Send application command failed.

35.3.2 enum _sdspi_card_flag

Enumerator

kSDSPI_SupportHighCapacityFlag Card is high capacity.
kSDSPI_SupportSdhcFlag Card is SDHC.
kSDSPI_SupportSdxcFlag Card is SDXC.
kSDSPI_SupportSdscFlag Card is SDSC.

35.3.3 enum sdspi_response_type_t

Enumerator

kSDSPI_ResponseTypeR1 Response 1.
kSDSPI_ResponseTypeR1b Response 1 with busy.
kSDSPI_ResponseTypeR2 Response 2.
kSDSPI_ResponseTypeR3 Response 3.
kSDSPI_ResponseTypeR7 Response 7.

35.4 Function Documentation

35.4.1 status_t SDSPI_Init (sdspi_card_t * *card*)

This function initializes the card on a specific SPI instance.

Parameters

card	Card descriptor

Return values

kStatus_SDSPI_SetFrequencyFailed	Set frequency failed.
kStatus_SDSPI_GoIdleFailed	Go idle failed.
kStatus_SDSPI_SendInterfaceConditionFailed	Send interface condition failed.

kStatus_SDSPI_SendOperationCondition-Failed	Send operation condition failed.
kStatus_Timeout	Send command timeout.
kStatus_SDSPI_Not-SupportYet	Not support yet.
kStatus_SDSPI_ReadOcr-Failed	Read OCR failed.
kStatus_SDSPI_SetBlock-SizeFailed	Set block size failed.
kStatus_SDSPI_SendCsd-Failed	Send CSD failed.
kStatus_SDSPI_SendCid-Failed	Send CID failed.
kStatus_Success	Operate successfully.

35.4.2 void SDSPI_Deinit (sdspi_card_t * *card*)

This function deinitializes the specific card.

Parameters

card	Card descriptor

35.4.3 bool SDSPI_CheckReadOnly (sdspi_card_t * *card*)

This function checks if the card is write-protected via CSD register.

Parameters

card	Card descriptor.

Return values

true	Card is read only.
false	Card isn't read only.

35.4.4 status_t SDSPI_ReadBlocks (sdspi_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function reads blocks from specific card.

Parameters

card	Card descriptor.
buffer	the buffer to hold the data read from card
startBlock	the start block index
blockCount	the number of blocks to read

Return values

kStatus_SDSPI_SendCommandFailed	Send command failed.
kStatus_SDSPI_ReadFailed	Read data failed.
kStatus_SDSPI_StopTransmissionFailed	Stop transmission failed.
kStatus_Success	Operate successfully.

35.4.5 status_t SDSPI_WriteBlocks (sdspi_card_t * *card*, uint8_t * *buffer*, uint32_t *startBlock*, uint32_t *blockCount*)

This function writes blocks to specific card

Parameters

card	Card descriptor.
buffer	the buffer holding the data to be written to the card

startBlock	the start block index
blockCount	the number of blocks to write

Return values

kStatus_SDSPI_WriteProtected	Card is write protected.
kStatus_SDSPI_SendCommandFailed	Send command failed.
kStatus_SDSPI_ResponseError	Response is error.
kStatus_SDSPI_WriteFailed	Write data failed.
kStatus_SDSPI_ExchangeFailed	Exchange data over SPI failed.
kStatus_SDSPI_WaitReadyFailed	Wait card to be ready status failed.
kStatus_Success	Operate successfully.

Chapter 36
Debug Console

36.1 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.

36.2 Function groups

36.2.1 Initialization

To initialize the debug console, call the DbgConsole_Init() function with these parameters. This function automatically enables the module and the clock.

```
/*
 * @brief Initializes the the peripheral used to debug messages.
 *
 * @param baseAddr      Indicates which address of the peripheral is used to send debug messages.
 * @param baudRate      The desired baud rate in bits per second.
 * @param device        Low level device type for the debug console, can be one of:
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_UART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPUART,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_LPSCI,
 *                      @arg DEBUG_CONSOLE_DEVICE_TYPE_USBCDC.
 * @param clkSrcFreq    Frequency of peripheral source clock.
 *
 * @return              Whether initialization was successful or not.
 */
status_t DbgConsole_Init(uint32_t baseAddr, uint32_t baudRate, uint8_t device, uint32_t clkSrcFreq)
```

Selects the supported debug console hardware device type, such as

```
DEBUG_CONSOLE_DEVICE_TYPE_NONE
DEBUG_CONSOLE_DEVICE_TYPE_LPSCI
DEBUG_CONSOLE_DEVICE_TYPE_UART
DEBUG_CONSOLE_DEVICE_TYPE_LPUART
DEBUG_CONSOLE_DEVICE_TYPE_USBCDC
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the debug_console_state_t structure, such as shown here.

```
typedef struct DebugConsoleState
{
    uint8_t                     type;
    void*                       base;
    debug_console_ops_t         ops;
} debug_console_state_t;
```

This example shows how to call the DbgConsole_Init() given the user configuration structure.

```
uint32_t uartClkSrcFreq = CLOCK_GetFreq(BOARD_DEBUG_UART_CLKSRC);

DbgConsole_Init(BOARD_DEBUG_UART_BASEADDR, BOARD_DEBUG_UART_BAUDRATE, DEBUG_CONSOLE_DEVICE_TYPE_UART,
      uartClkSrcFreq);
```

36.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " %[flags][width][.precision][length]specifier", which is explained below

flags	Description
-	Left-justified within the given field width. Right-justified is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is written, a blank space is inserted before the value.
#	Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed.
0	Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).

Width	Description
(number)	A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

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

length	Description
Do not support	

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

- Support a format specifier for SCANF following this prototype " %[∗][width][length]specifier", which is explained below

∗	Description
An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument.	

width	Description
This specifies the maximum number of characters to be read in the current reading operation.	

length	Description
hh	The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).
h	The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).
l	The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
ll	The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.
L	The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).
j or z or t	Not supported

specifier	Qualifying Input	Type of argument
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	char ∗

specifier	Qualifying Input	Type of argument
i	Integer: : Number optionally preceded with a + or - sign	int ∗
d	Decimal integer: Number optionally preceded with a + or - sign	int ∗
a, A, e, E, f, F, g, G	Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	float ∗
o	Octal Integer:	int ∗
s	String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).	char ∗
u	Unsigned decimal integer.	unsigned int ∗

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(const char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE    /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF                DbgConsole_Printf
#define SCANF                 DbgConsole_Scanf
#define PUTCHAR               DbgConsole_Putchar
#define GETCHAR               DbgConsole_Getchar
#else                   /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF                printf
#define SCANF                 scanf
#define PUTCHAR               putchar
#define GETCHAR               getchar
#endif /* SDK_DEBUGCONSOLE */
```

36.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

Typical use case

Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalents 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

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

Some examples use the SCANF function

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

Print out failure messages using KSDK __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

- Semihosting

36.4 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.

36.4.1 Guide Semihosting for IAR

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

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. Start the project by choosing Project>Download and Debug.
4. Choose View>Terminal I/O to display the output from the I/O operations.

36.4.2 Guide Semihosting for Keil µVision

NOTE: Keil supports Semihosting only for Cortex-M3/Cortex-M4 cores.

Step 1: Prepare code

Remove function fputc and fgetc is used to support KEIL in "fsl_debug_console.c" and add the following code to project.

```
#pragma import(__use_no_semihosting_swi)

volatile int ITM_RxBuffer = ITM_RXBUFFER_EMPTY;       /* used for Debug Input */
```

```
struct __FILE
{
    int handle;
};
FILE __stdout;
FILE __stdin;

int fputc(int ch, FILE *f)
{
    return (ITM_SendChar(ch));
}

int fgetc(FILE *f)
{ /* blocking */
    while (ITM_CheckChar() != 1)
        ;
    return (ITM_ReceiveChar());
}

int ferror(FILE *f)
{
    /* Your implementation of ferror */
    return EOF;
}

void _ttywrch(int ch)
{
    ITM_SendChar(ch);
}

void _sys_exit(int return_code)
{
label:
    goto label; /* endless loop */
}
```

Step 2: Setting up the environment

1. In menu bar, choose Project>Options for target or using Alt+F7 or click.
2. Select "Target" tab and not select "Use MicroLIB".
3. Select “Debug” tab, select “J-Link/J-Trace Cortex” and click “Setting button”.
4. Select “Debug” tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click OK.

Step 3: Building the project

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

Step 4: Building 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.

36.4.3 Guide Semihosting for KDS

NOTE: After the setting use "printf" for debugging.

Step 1: Setting up the environment

1. In menu bar, choose Project>Properties>C/C++ Build>Settings>Tool Settings.
2. Select “Libraries” on “Cross ARM C Linker” and delete “nosys”.
3. Select “Miscellaneous” on "Cross ARM C Linker”, add “-specs=rdimon.specs” to “Other link flages” and tick “Use newlib-nano”, and click OK.

Step 2: Building the project

1. In menu bar, choose Project>Build Project.

Step 3: Starting semihosting

1. In Debug configurations, choose "Startup" tab, tick “Enable semihosting and Telnet”. Press “Apply” and “Debug”.
2. After clicking Debug, the Window is displayed same as below. Run line by line to see the result in the Console Window.

36.4.4 Guide Semihosting for ATL

NOTE: J-Link has to be used to enable semihosting.

Step 1: Prepare code

Add the following code to the project.

```
int _write(int file, char *ptr, int len)
{
  /* Implement your write code here. This is used by puts and printf. */
  int i=0;
  for(i=0 ; i<len ; i++)
    ITM_SendChar((*ptr++));
  return len;
}
```

Step 2: Setting up the environment

1. In menu bar, choose Debug Configurations. In tab "Embedded C/C++ Aplication" choose "-Semihosting_ATL_xxx debug J-Link".
2. In tab "Debugger" set up as follows.
   - JTAG mode must be selected

- SWV tracing must be enabled
- Enter the Core Clock frequency, which is hardware board-specific.
- Enter the desired SWO Clock frequency. The latter depends on the JTAG Probe and must be a multiple of the Core Clock value.

3. Click "Apply" and "Debug".

Step 3: Starting semihosting

1. In the Views menu, expand the submenu SWV and open the docking view “SWV Console". 2. Open the SWV settings panel by clicking the "Configure Serial Wire Viewer" button in the SWV Console view toolbar. 3. Configure the data ports to be traced by enabling the ITM channel 0 check-box in the ITM stimulus ports group: Choose "EXETRC: Trace Exceptions" and In tab "ITM Stimulus Ports" choose "Enable Port" 0. Then click "OK".
2. It is recommended not to enable other SWV trace functionalities at the same time because this may over use the SWO pin causing packet loss due to a limited bandwidth (certain other SWV tracing capabilities can send a lot of data at very high-speed). Save the SWV configuration by clicking the OK button. The configuration is saved with other debug configurations and remains effective until changed.
3. Press the red Start/Stop Trace button to send the SWV configuration to the target board to enable SWV trace recoding. The board does not send any SWV packages until it is properly configured. The SWV Configuration must be present, if the configuration registers on the target board are reset. Also, tracing does not start until the target starts to execute.
4. Start the target execution again by pressing the green Resume Debug button.
5. The SWV console now shows the printf() output.

36.4.5 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_DEBU-

G} --specs=rdimon.specs ")
Remove
target_link_libraries(semihosting_ARMGCC.elf debug nosys)

2. Run "build_debug.bat" to build project

Step 3: Starting semihosting

(a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\twrk64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

(b) After the setting, press "enter". The PuTTY window now shows the printf() output.

Chapter 37
Notification Framework

37.1 Overview

This section describes the programming interface of the Notifier driver.

37.2 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.

37.3 Data Structure Documentation

37.3.1 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.

37.3.1.0.0.14 Field Documentation

37.3.1.0.0.14.1 notifier_user_config_t* notifier_notification_block_t::targetConfig

37.3.1.0.0.14.2 notifier_policy_t notifier_notification_block_t::policy

37.3.1.0.0.14.3 notifier_notification_type_t notifier_notification_block_t::notifyType

37.3.2 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.

37.3.2.0.0.15 Field Documentation

37.3.2.0.0.15.1 notifier_callback_t notifier_callback_config_t::callback

37.3.2.0.0.15.2 notifier_callback_type_t notifier_callback_config_t::callbackType

37.3.2.0.0.15.3 void* notifier_callback_config_t::callbackData

37.3.3 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.

37.3.3.0.0.16 Field Documentation

37.3.3.0.0.16.1 notifier_user_config_t∗∗ notifier_handle_t::configsTable

37.3.3.0.0.16.2 uint8_t notifier_handle_t::configsNumber

37.3.3.0.0.16.3 notifier_callback_config_t∗ notifier_handle_t::callbacksTable

37.3.3.0.0.16.4 uint8_t notifier_handle_t::callbacksNumber

37.3.3.0.0.16.5 uint8_t notifier_handle_t::errorCallbackIndex

37.3.3.0.0.16.6 uint8_t notifier_handle_t::currentConfigIndex

37.3.3.0.0.16.7 notifier_user_function_t notifier_handle_t::userFunction

37.3.3.0.0.16.8 void∗ notifier_handle_t::userData

37.4 Typedef Documentation

37.4.1 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.

37.4.2 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

targetConfig	target Configuration.
userData	Refers to other specific data passed to user function.

Returns

An error code or kStatus_Success.

37.4.3 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

notify	Notification block.
data	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.

37.5 Enumeration Type Documentation

37.5.1 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.

37.5.2 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.

37.5.3 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.

37.5.4 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.

37.6 Function Documentation

37.6.1 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

notifierHandle	A pointer to the notifier handle.
configs	A pointer to an array with references to all configurations which is handled by the Notifier.
configsNumber	Number of configurations. Size of the configuration array.
callbacks	A pointer to an array of callback configurations. If there are no callbacks to register during Notifier initialization, use NULL value.
callbacks-Number	Number of registered callbacks. Size of the callbacks array.
userFunction	User function.
userData	User data passed to user function.

Returns

An error Code or kStatus_Success.

37.6.2 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

notifierHandle	pointer to notifier handle
configIndex	Index of the target configuration.
policy	Transaction policy, kNOTIFIER_PolicyAgreement or kNOTIFIER_PolicyForcible.

Returns

An error code or kStatus_Success.

37.6.3 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

notifierHandle	Pointer to the notifier handle

Returns

Callback Index of the last failed callback or value equal to callbacks count.

Chapter 38
Shell

38.1 Overview

This part describes the programming interface of the Shell middleware. Shell controls MCUs by commands via the specified communication peripheral based on the debug console driver.

38.2 Function groups

38.2.1 Initialization

To initialize the Shell middleware, call the SHELL_Init() function with these parameters. This function automatically enables the middleware.

```
void SHELL_Init(p_shell_context_t context, send_data_cb_t send_cb,
      recv_data_cb_t recv_cb, 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(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
```

38.2.2 Advanced Feature

- Support to get a character from standard input devices.

  ```
  static uint8_t GetChar(p_shell_context_t context);
  ```

Commands	Description
Help	Lists all commands which are supported by Shell.
Exit	Exits the Shell program.
strCompare	Compares the two input strings.

Input character	Description
A	Gets the latest command in the history.
B	Gets the first command in the history.
C	Replaces one character at the right of the pointer.

Input character	Description
D	Replaces one character at the left of the pointer.
	Run AutoComplete function
	Run cmdProcess function
	Clears a command.

38.2.3 Shell Operation

```
SHELL_Init(&user_context, SHELL_SendDataCallback, SHELL_ReceiveDataCallback, "SHELL>> ");
SHELL_Main(&user_context);
```

Data Structures

- struct p_shell_context_t
  Data structure for Shell environment. More...
- struct shell_command_context_t
  User command data structure. More...
- struct shell_command_context_list_t
  Structure list command. More...

Macros

- #define SHELL_USE_HISTORY (0U)
  Macro to set on/off history feature.
- #define SHELL_SEARCH_IN_HIST (1U)
  Macro to set on/off history feature.
- #define SHELL_USE_FILE_STREAM (0U)
  Macro to select method stream.
- #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_HIST_MAX (3U)
  Macro to set maximum count of history commands.
- #define SHELL_MAX_CMD (20U)
  Macro to set maximum count of commands.
- #define SHELL_OPTIONAL_PARAMS (0xFF)
  Macro to bypass arguments check.

Typedefs

- typedef void(∗ send_data_cb_t)(uint8_t ∗buf, uint32_t len)
  Shell user send data callback prototype.
- typedef void(∗ recv_data_cb_t)(uint8_t ∗buf, uint32_t len)

Shell user receiver data callback prototype.

- typedef int(∗ printf_data_t)(const char ∗format,...)
  Shell user printf data prototype.
- typedef int32_t(∗ cmd_function_t)(p_shell_context_t context, int32_t argc, char ∗∗argv)
  User command function prototype.

Enumerations

- enum fun_key_status_t {
  kSHELL_Normal = 0U,
  kSHELL_Special = 1U,
  kSHELL_Function = 2U }
  A type for the handle special key.

Shell functional operation

- void SHELL_Init (p_shell_context_t context, send_data_cb_t send_cb, recv_data_cb_t recv_cb, printf_data_t shell_printf, char ∗prompt)
  Enables the clock gate and configures the Shell module according to the configuration structure.
- int32_t SHELL_RegisterCommand (const shell_command_context_t ∗command_context)
  Shell register command.
- int32_t SHELL_Main (p_shell_context_t context)
  Main loop for Shell.

38.3 Data Structure Documentation

38.3.1 struct shell_context_struct

Data Fields

- char ∗ prompt
  Prompt string.
- enum _fun_key_status stat
  Special key status.
- char line [SHELL_BUFFER_SIZE]
  Consult buffer.
- uint8_t cmd_num
  Number of user commands.
- uint8_t l_pos
  Total line position.
- uint8_t c_pos
  Current line position.
- send_data_cb_t send_data_func
  Send data interface operation.
- recv_data_cb_t recv_data_func
  Receive data interface operation.
- uint16_t hist_current
  Current history command in hist buff.
- uint16_t hist_count

Total history command in hist buff.
- char hist_buf [SHELL_HIST_MAX][SHELL_BUFFER_SIZE]
  History buffer.
- bool exit
  Exit Flag.

38.3.2 struct shell_command_context_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.

38.3.2.0.0.17 Field Documentation

38.3.2.0.0.17.1 const char* shell_command_context_t::pcCommand

For example "help". It must be all lower case.

38.3.2.0.0.17.2 char* shell_command_context_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".

38.3.2.0.0.17.3 const cmd_function_t shell_command_context_t::pFuncCallBack

38.3.2.0.0.17.4 uint8_t shell_command_context_t::cExpectedNumberOfParameters

38.3.3 struct shell_command_context_list_t

Data Fields

- const shell_command_context_t * CommandList [SHELL_MAX_CMD]
  The command table list.
- uint8_t numberOfCommandInList
  The total command in list.

38.4 Macro Definition Documentation

38.4.1 #define SHELL_USE_HISTORY (0U)

38.4.2 #define SHELL_SEARCH_IN_HIST (1U)

38.4.3 #define SHELL_USE_FILE_STREAM (0U)

38.4.4 #define SHELL_AUTO_COMPLETE (1U)

38.4.5 #define SHELL_BUFFER_SIZE (64U)

38.4.6 #define SHELL_MAX_ARGS (8U)

38.4.7 #define SHELL_HIST_MAX (3U)

38.4.8 #define SHELL_MAX_CMD (20U)

38.5 Typedef Documentation

38.5.1 typedef void(∗ send_data_cb_t)(uint8_t ∗buf, uint32_t len)

38.5.2 typedef void(∗ recv_data_cb_t)(uint8_t ∗buf, uint32_t len)

38.5.3 typedef int(∗ printf_data_t)(const char ∗format,...)

38.5.4 typedef int32_t(∗ cmd_function_t)(p_shell_context_t context, int32_t argc, char ∗∗argv)

38.6 Enumeration Type Documentation

38.6.1 enum fun_key_status_t

Enumerator

kSHELL_Normal Normal key.
kSHELL_Special Special key.
kSHELL_Function Function key.

38.7 Function Documentation

38.7.1 void SHELL_Init (p_shell_context_t *context*, send_data_cb_t *send_cb*, recv_data_cb_t *recv_cb*, printf_data_t *shell_printf*, 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 middleware Shell and how to call the SHELL_Init function by passing in these parameters. This is an example.

```
*   shell_context_struct user_context;
*   SHELL_Init(&user_context, SendDataFunc, ReceiveDataFunc, "SHELL>> ");
*
```

Parameters

context	The pointer to the Shell environment and runtime states.
send_cb	The pointer to call back send data function.
recv_cb	The pointer to call back receive data function.
prompt	The string prompt of Shell

38.7.2 int32_t SHELL_RegisterCommand (const shell_command_context_t * *command_context*)

Parameters

command_-context	The pointer to the command data structure.

Returns

-1 if error or 0 if success

38.7.3 int32_t SHELL_Main (p_shell_context_t *context*)

Main loop for Shell; After this function is called, Shell begins to initialize the basic variables and starts to work.

Parameters

context	The pointer to the Shell environment and runtime states.

Returns

This function does not return until Shell command exit was called.

How to Reach Us:

Home Page:
nxp.com

Web Support:
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 reserves the right to make changes without further notice to any products herein. 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, Freescale, the Freescale logo, Kinetis, Processor Expert are trademarks of NXP B.V. Tower is a trademark of NXP. All other product or service names are the property of their respective owners. ARM, ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Document Number: MCUXSDKKW35AAPIRM
Rev. 0
Jun 2017