

Kinetis SDK API Reference Manual

Freescale Semiconductor, Inc.

Document Number: KSDKAPIRM
Rev. 1.1.0
Oct 2014



Contents

Chapter Introduction

Chapter Architectural Overview

Chapter 16-bit SAR Analog-to-Digital Converter (ADC16)

3.1	Overview	13
3.2	ADC16 HAL Driver	14
3.2.1	Overview	14
3.2.2	Enumeration Type Documentation	16
3.2.3	Function Documentation	18
3.3	ADC16 Peripheral Driver	26
3.3.1	Overview	26
3.3.2	ADC16 Driver model building	26
3.3.3	ADC16 Initialization	26
3.3.4	ADC16 Call diagram	26
3.3.5	Data Structure Documentation	33
3.3.6	Enumeration Type Documentation	35
3.3.7	Function Documentation	36
3.3.8	Variable Documentation	42

Chapter Comparator (CMP)

4.1	Overview	43
4.2	CMP HAL Driver	44
4.2.1	Overview	44
4.2.2	Enumeration Type Documentation	46
4.2.3	Function Documentation	47
4.3	CMP Peripheral Driver	57
4.3.1	Overview	57
4.3.2	CMP Driver model building	57
4.3.3	CMP Call diagram	57
4.3.4	Data Structure Documentation	61
4.3.5	Enumeration Type Documentation	63

Contents

Section Number	Title	Page Number
4.3.6	Function Documentation	64
4.3.7	Variable Documentation	69

Chapter Watchdog Timer

5.1	Overview	71
5.2	COP HAL driver	72
5.2.1	Overview	72
5.2.2	Data Structure Documentation	73
5.2.3	Enumeration Type Documentation	73
5.2.4	Function Documentation	73

Chapter 12-bit Cyclic Analog-to-Digital Converter (CADC)

6.1	Overview	77
6.2	CADC HAL Driver	78
6.2.1	Overview	78
6.2.2	Enumeration Type Documentation	81
6.2.3	Function Documentation	84
6.3	CADC Peripheral Driver	101
6.3.1	Overview	101
6.3.2	CADC Driver model building	101
6.3.3	CADC Work mode	101
6.3.4	CADC Interrupt	102
6.3.5	CADC Call diagram	102
6.3.6	Data Structure Documentation	106
6.3.7	Enumeration Type Documentation	109
6.3.8	Function Documentation	110
6.3.9	Variable Documentation	116

Chapter Digital-to-Analog Converter (DAC)

7.1	Overview	117
7.2	DAC HAL Driver	118
7.2.1	Overview	118
7.2.2	Enumeration Type Documentation	119
7.2.3	Function Documentation	120
7.3	DAC Peripheral Driver	128
7.3.1	Overview	128
7.3.2	DAC Initialization	128

Contents

Section Number	Title	Page Number
7.3.3	DAC Model building	128
7.3.4	DAC Call diagram	129
7.3.5	Data Structure Documentation	136
7.3.6	Enumeration Type Documentation	137
7.3.7	Function Documentation	138
7.3.8	Variable Documentation	141

Chapter Direct Memory Access (DMA)

8.1	Overview	143
8.2	DMA HAL driver	144
8.2.1	Overview	144
8.2.2	Data Structure Documentation	146
8.2.3	Enumeration Type Documentation	146
8.2.4	Function Documentation	147
8.3	DMAMUX HAL driver	156
8.3.1	Overview	156
8.3.2	Enumeration Type Documentation	156
8.3.3	Function Documentation	156
8.4	DMA driver	158
8.4.1	Overview	158
8.4.2	Data Structure Documentation	159
8.4.3	Typedef Documentation	160
8.4.4	Enumeration Type Documentation	160
8.4.5	Function Documentation	160
8.4.6	Variable Documentation	164
8.5	DMA request	165
8.5.1	DMA Initialization	165
8.5.2	DMA Channel concept	165
8.5.3	DMA request concept	165
8.5.4	DMA Memory allocation	165
8.5.5	DMA Call diagram	165

Chapter Serial Peripheral Interface (DSPI)

9.1	Overview	167
9.2	DSPI HAL driver	168
9.2.1	Overview	168
9.2.2	Data Structure Documentation	173
9.2.3	Enumeration Type Documentation	174

Contents

Section Number	Title	Page Number
9.2.4	Function Documentation	177
9.3	DSPI Master Driver	194
9.3.1	Overview	194
9.3.2	DSPI Introduction	194
9.3.3	DSPI Run-time state structures	195
9.3.4	DSPI User configuration structures	195
9.3.5	DSPI Device structures	195
9.3.6	DSPI Initialization	196
9.3.7	DSPI Transfers	198
9.3.8	DSPI De-initialization	200
9.3.9	Data Structure Documentation	202
9.3.10	Function Documentation	206
9.3.11	Variable Documentation	218
9.4	DSPI Slave Driver	219
9.4.1	Overview	219
9.4.2	DSPI Runtime state of the DSPI slave driver	220
9.4.3	DSPI User configuration structures	220
9.4.4	DSPI Setup and Initialization	220
9.4.5	DSPI Blocking and non-blocking	221
9.4.6	DSPI De-initialization	222
9.4.7	Data Structure Documentation	224
9.4.8	Function Documentation	226
9.4.9	Variable Documentation	233
9.5	DSPI Shared IRQ Driver	234
9.5.1	Overview	234
9.5.2	Function Documentation	234
 Chapter Enhanced Direct Memory Access (eDMA)		
10.1	Overview	235
10.2	eDMA HAL driver	236
10.2.1	Overview	236
10.2.2	Data Structure Documentation	241
10.2.3	Enumeration Type Documentation	243
10.2.4	Function Documentation	244
10.3	eDMA Peripheral driver	270
10.3.1	Overview	270
10.3.2	eDMA Initialization	270
10.3.3	eDMA Channel concept	270
10.3.4	DMA request concept	270

Contents

Section Number	Title	Page Number
10.3.5	eDMA Memory allocation and alignment	270
10.3.6	eDMA Call diagram	271
10.3.7	eDMA Extend the driver	272
10.3.8	Data Structure Documentation	275
10.3.9	Macro Definition Documentation	276
10.3.10	Typedef Documentation	277
10.3.11	Enumeration Type Documentation	277
10.3.12	Function Documentation	278
10.3.13	Variable Documentation	287
10.4	eDMA request	288
10.4.1	Overview	288
10.4.2	Enumeration Type Documentation	288
Chapter	Ethernet MAC (ENET)	
11.1	Overview	289
11.2	ENET HAL driver	290
11.2.1	Overview	290
11.2.2	Data Structure Documentation	301
11.2.3	Macro Definition Documentation	307
11.2.4	Enumeration Type Documentation	307
11.2.5	Function Documentation	314
Chapter	external Watchdog Timer (EWM)	
12.1	Overview	371
12.2	EWM HAL driver	372
12.2.1	Overview	372
12.2.2	Data Structure Documentation	373
12.2.3	Enumeration Type Documentation	373
12.2.4	Function Documentation	373
Chapter	C90TFS Flash Driver	
13.1	Overview	379
13.2	Data Structure Documentation	387
13.2.1	struct FLASH_SSD_CONFIG	387
13.3	Macro Definition Documentation	388
13.3.1	BYTE2WORD	388
13.3.2	WORD2BYTE	388

Contents

Section Number	Title	Page Number
13.3.3	SET_FLASH_INT_BITS	388
13.3.4	GET_FLASH_INT_BITS	388
13.3.5	FTFx_ERR_MGSTAT0	388
13.3.6	FTFx_ERR_PVIOL	389
13.3.7	FTFx_ERR_ACCERR	389
13.3.8	FTFx_ERR_CHANGEPROT	389
13.3.9	FTFx_ERR_NOEEE	389
13.3.10	FTFx_ERR_EFLASHONLY	390
13.3.11	FTFx_ERR_RAMRDY	390
13.3.12	FTFx_ERR_RANGE	390
13.3.13	FTFx_ERR_SIZE	390
13.4	Function Documentation	390
13.4.1	RelocateFunction	390
13.4.2	FlashInit	391
13.4.3	FlashCommandSequence	391
13.4.4	PFlashGetProtection	392
13.4.5	PFlashSetProtection	392
13.4.6	FlashGetSecurityState	393
13.4.7	FlashSecurityBypass	393
13.4.8	FlashEraseAllBlock	394
13.4.9	FlashVerifyAllBlock	395
13.4.10	FlashEraseSector	395
13.4.11	FlashVerifySection	396
13.4.12	FlashEraseSuspend	397
13.4.13	FlashEraseResume	398
13.4.14	FlashReadOnce	398
13.4.15	FlashProgramOnce	399
13.4.16	FlashReadResource	400
13.4.17	FlashProgram	401
13.4.18	FlashProgramCheck	401
13.4.19	FlashCheckSum	402
13.4.20	FlashProgramSection	403
13.4.21	FlashEraseBlock	403
13.4.22	FlashVerifyBlock	404
Chapter	Controller Area Network (FlexCAN)	
14.1	Overview	405
14.2	FlexCAN HAL driver	406
14.2.1	Overview	406
14.2.2	Data Structure Documentation	411
14.2.3	Enumeration Type Documentation	412
14.2.4	Function Documentation	415

Contents

Section Number	Title	Page Number
14.3	FlexCAN Driver	433
14.3.1	Overview	433
14.3.2	FlexCAN Overview	433
14.3.3	FlexCAN Initialization	433
14.3.4	FlexCAN Module timing	433
14.3.5	FlexCAN Transfers	434
14.3.6	Data Structure Documentation	436
14.3.7	Function Documentation	437
14.3.8	Variable Documentation	442
 Chapter FlexTimer (FTM)		
15.1	Overview	443
15.2	FlexTimer HAL driver	444
15.2.1	Overview	444
15.2.2	Data Structure Documentation	450
15.2.3	Macro Definition Documentation	451
15.2.4	Enumeration Type Documentation	451
15.2.5	Function Documentation	451
15.3	FlexTimer Peripheral Driver	484
15.3.1	Overview	484
15.3.2	FlexTimer Overview	484
15.3.3	FlexTimer Initialization	484
15.3.4	FlexTimer Generate a PWM signal	484
15.3.5	Data Structure Documentation	486
15.3.6	Function Documentation	486
15.3.7	Variable Documentation	490
 Chapter General Purpose Input/Output (GPIO)		
16.1	Overview	491
16.2	GPIO HAL driver	492
16.2.1	Overview	492
16.2.2	Enumeration Type Documentation	493
16.2.3	Function Documentation	493
16.3	GPIO Peripheral driver	499
16.3.1	Overview	499
16.3.2	GPIO Pin Definitions	499
16.3.3	GPIO Initialization	499
16.3.4	Output Operations	500
16.3.5	Input Operations	501

Contents

Section Number	Title	Page Number
16.3.6	GPIO Interrupt	501
16.3.7	Data Structure Documentation	503
16.3.8	Macro Definition Documentation	504
16.3.9	Function Documentation	504
16.3.10	Variable Documentation	508

Chapter Inter-Integrated Circuit (I2C)

17.1	Overview	509
17.2	I2C HAL driver	510
17.2.1	Overview	510
17.2.2	I2C ICR Table	510
17.2.3	I2C Clock rate formulas	510
17.2.4	Enumeration Type Documentation	513
17.2.5	Function Documentation	514
17.3	I2C Slave peripheral driver	526
17.3.1	Overview	526
17.3.2	Data Structure Documentation	527
17.3.3	Enumeration Type Documentation	529
17.3.4	Function Documentation	530
17.3.5	Variable Documentation	535
17.4	I2C Master peripheral	536
17.4.1	Overview	536
17.4.2	I2C Initialization	536
17.4.3	I2C Data Transactions	536
17.4.4	Data Structure Documentation	537
17.4.5	Function Documentation	538
17.4.6	Variable Documentation	544

Chapter Universal Asynchronous Receiver/Transmitter (UART)

18.1	Overview	545
18.2	UART HAL driver	546
18.2.1	Overview	546
18.2.2	Enumeration Type Documentation	550
18.2.3	Function Documentation	553
18.3	UART Peripheral driver	567
18.3.1	Overview	567
18.3.2	UART Device structures	567
18.3.3	UART User configuration structures	567

Contents

Section Number	Title	Page Number
18.3.4	UART Initialization	567
18.3.5	UART Transfers	568
18.3.6	UART Device structures	569
18.3.7	UART User configuration structures	569
18.3.8	UART Initialization	569
18.3.9	UART Transfers	570
18.3.10	Data Structure Documentation	573
18.3.11	Function Documentation	576
18.3.12	Variable Documentation	585

Chapter Low Power Timer (LPTMR)

19.1	Overview	587
19.2	LPTMR HAL driver	588
19.2.1	Overview	588
19.2.2	Enumeration Type Documentation	590
19.2.3	Function Documentation	591
19.3	LPTMR Peripheral driver	599
19.3.1	Overview	599
19.3.2	LPTMR Initialization	599
19.3.3	LPTMR Interrupt	599
19.3.4	Data Structure Documentation	601
19.3.5	Function Documentation	602
19.3.6	Variable Documentation	605

Chapter Low Power Universal Asynchronous Receiver/Transmitter (LPUART)

20.1	Overview	607
20.2	LPUART HAL driver	608
20.2.1	Overview	608
20.2.2	Data Structure Documentation	614
20.2.3	Enumeration Type Documentation	614
20.2.4	Function Documentation	618
20.3	LPUART Peripheral driver	636
20.3.1	Overview	636
20.3.2	Data Structure Documentation	638
20.3.3	Typedef Documentation	641
20.3.4	Function Documentation	641
20.3.5	Variable Documentation	650
20.4	LPUART Type Definitions	651

Contents

Section Number	Title	Page Number
20.4.1	LPUART Overview	651
20.4.2	LPUART Device structures	651
20.4.3	LPUART Initialization	651
20.4.4	LPUART Transfers	652
Chapter Memory Protection Unit (MPU)		
21.1	Overview	653
21.2	MPU HAL driver	654
21.2.1	Overview	654
21.2.2	Enumeration Type Documentation	660
21.2.3	Function Documentation	663
21.3	MPU Peripheral driver	687
21.3.1	Overview	687
21.3.2	MPU Initialization	687
21.3.3	MPU Interrupt	688
21.3.4	Data Structure Documentation	689
21.3.5	Function Documentation	690
21.3.6	Variable Documentation	693
Chapter Programmable Delay Block (PDB)		
22.1	Overview	695
22.2	PDB HAL driver	696
22.2.1	Overview	696
22.2.2	Enumeration Type Documentation	699
22.2.3	Function Documentation	700
22.3	PDB Peripheral driver	714
22.3.1	Overview	714
22.3.2	PDB Driver model building	714
22.3.3	PDB Initialization	714
22.3.4	PDB Call diagram	714
22.3.5	Data Structure Documentation	722
22.3.6	Enumeration Type Documentation	724
22.3.7	Function Documentation	724
22.3.8	Variable Documentation	730
Chapter Periodic Interrupt Timer (PIT)		
23.1	Overview	731

Contents

Section Number	Title	Page Number
23.2	PIT HAL driver	732
23.2.1	Overview	732
23.2.2	Function Documentation	733
23.3	PIT Peripheral driver	739
23.3.1	Overview	739
23.3.2	PIT Initialization	739
23.3.3	PIT Timer Period	739
23.3.4	PIT Timer Operation	740
23.3.5	Data Structure Documentation	741
23.3.6	Function Documentation	741
23.3.7	Variable Documentation	745
Chapter	Random Number Generator Accelerator (RNGA)	
24.1	Overview	747
24.2	RNGA HAL driver	748
24.2.1	Overview	748
24.2.2	Enumeration Type Documentation	749
24.2.3	Function Documentation	750
24.3	RNGA Peripheral Driver	757
24.3.1	Overview	757
24.3.2	RNGA Initialization	757
24.3.3	RNGA Set/Get Working Mode	757
24.3.4	Get random data from RNGA	757
24.3.5	Seed RNGA	757
24.3.6	RNGA interrupt	757
24.3.7	Data Structure Documentation	758
24.3.8	Function Documentation	759
24.3.9	Variable Documentation	761
Chapter	Real Time Clock (RTC)	
25.1	Overview	763
25.2	RTC HAL driver	764
25.2.1	Overview	764
25.2.2	RTC Initialization	764
25.2.3	RTC Setting and reading the RTC time	764
25.2.4	RTC Setting and reading the Alarm	764
25.2.5	Data Structure Documentation	768
25.2.6	Function Documentation	768

Contents

Section Number	Title	Page Number
25.3	RTC Peripheral Driver	793
25.3.1	Overview	793
25.3.2	RTC Peripheral Driver Initialization	793
25.3.3	RTCS Setting and reading the RTC time	793
25.3.4	RTC Triggering an Alarm	793
25.3.5	RTC Repeated alarm	794
25.3.6	RTC Enable and Disable Alarm Interrupts	795
25.3.7	RTC Interrupt handler	795
25.3.8	Data Structure Documentation	797
25.3.9	Function Documentation	797
25.3.10	Variable Documentation	804
 Chapter Synchronous Audio Interface (SAI)		
26.1	Overview	805
26.2	SAI HAL driver	806
26.2.1	Overview	806
26.2.2	Enumeration Type Documentation	810
26.2.3	Function Documentation	812
26.3	SAI Peripheral driver	826
26.3.1	Overview	826
26.3.2	SAI Initialization	826
26.3.3	SAI Configuration	826
26.3.4	SAI Call diagram	826
26.3.5	Data Structure Documentation	829
26.3.6	Function Documentation	830
 Chapter Secured Digital Host Controller (SDHC)		
27.1	Overview	839
27.2	SDHC HAL	840
27.2.1	Overview	840
27.2.2	Function Documentation	843
27.3	SDHC Peripheral Driver	875
27.3.1	Overview	875
27.3.2	SDHC Initialization	875
27.3.3	SDHC Issuing a request to the card	875
27.3.4	Function Documentation	876
27.4	SDHC Data Types	879
27.4.1	Overview	879

Contents

Section Number	Title	Page Number
27.4.2	Data Structure Documentation	881
27.4.3	Enumeration Type Documentation	884
27.5	SDHC Standard Definition	887
27.5.1	Overview	887
27.6	SDHC Card Related Standard Definition	896
27.6.1	Overview	896
27.6.2	Enumeration Type Documentation	901
27.7	SDHC Card Definition	904
27.8	SDHC Card Driver	905
27.9	SD Card SPI Data Definition	906
27.10	SD Card SPI Driver	907
Chapter	Serial Peripheral Interface (SPI)	
28.1	Overview	909
28.2	SPI HAL driver	910
28.2.1	Overview	910
28.2.2	Enumeration Type Documentation	913
28.2.3	Function Documentation	916
28.3	SPI Master Peripheral Driver	926
28.3.1	Overview	926
28.3.2	SPI Run-time state structures	927
28.3.3	SPI Device structures	927
28.3.4	SPI Initialization	927
28.3.5	SPI Transfers	929
28.3.6	SPI De-initialization	930
28.3.7	Data Structure Documentation	933
28.3.8	Enumeration Type Documentation	935
28.3.9	Function Documentation	936
28.3.10	Variable Documentation	943
28.4	SPI Slave Peripheral Driver	944
28.4.1	Overview	944
28.4.2	SPI Overview	944
28.4.3	SPI Runtime state of the SPI slave driver	945
28.4.4	SPI User configuration structures	945
28.4.5	SPI Setup and Initialization	945
28.4.6	SPI Blocking and non-blocking	946

Contents

Section Number	Title	Page Number
28.4.7	SPI De-initialization	947
28.4.8	Data Structure Documentation	949
28.4.9	Function Documentation	952
28.5	Shared SPI Types	958
28.6	SPI Classes	959
Chapter	Timer/PWM Module (TPM)	
29.1	Overview	961
29.2	TPM HAL driver	962
29.2.1	Overview	962
29.2.2	Data Structure Documentation	964
29.2.3	Enumeration Type Documentation	964
29.2.4	Function Documentation	964
29.3	TPM Peripheral driver	977
29.3.1	Overview	977
29.3.2	TPM Initialization	977
29.3.3	TPM Generate a PWM signal	977
29.3.4	TPM Input Capture	977
29.3.5	TPM Output Compare	978
29.3.6	TPM Interrupt handler	978
29.3.7	Data Structure Documentation	980
29.3.8	Enumeration Type Documentation	980
29.3.9	Function Documentation	981
29.3.10	Variable Documentation	985
Chapter	Touch Sense Input (TSI)	
30.1	Overview	987
30.2	TSI HAL driver	988
30.2.1	Overview	988
30.2.2	Data Structure Documentation	999
30.2.3	Enumeration Type Documentation	1002
30.2.4	Function Documentation	1011
30.3	TSI Peripheral Driver	1040
30.3.1	Overview	1040
30.3.2	Data Structure Documentation	1044
30.3.3	Typedef Documentation	1046
30.3.4	Enumeration Type Documentation	1047
30.3.5	Function Documentation	1047

Contents

Section Number	Title	Page Number
30.3.6	Variable Documentation	1056
Chapter Watchdog Timer (WDOG)		
31.1	Overview	1057
31.2	WDOG HAL driver	1058
31.2.1	Overview	1058
31.2.2	Data Structure Documentation	1060
31.2.3	Enumeration Type Documentation	1060
31.2.4	Function Documentation	1060
31.3	WDOG Peripheral driver	1071
31.3.1	Overview	1071
31.3.2	WDOG Initialization	1071
31.3.3	WDOG Refresh	1071
31.3.4	WDOG Reset Count	1071
31.3.5	WDOG Reset System	1072
31.3.6	WDOG interrupt	1072
31.3.7	Data Structure Documentation	1073
31.3.8	Function Documentation	1073
Chapter Low-Leakage Wakeup Unit (LLWU)		
32.1	Overview	1075
32.2	Example	1075
32.3	Data Structure Documentation	1076
32.3.1	struct llwu_external_pin_filter_mode_t	1076
32.3.2	struct llwu_reset_enable_mode_t	1076
Chapter Multipurpose Clock Generator (MCG)		
33.1	Overview	1077
33.2	MCG HAL driver	1078
33.2.1	Overview	1078
33.2.2	OSC related APIs	1078
33.2.3	FLL reference clock source, divider, and output frequency	1078
33.2.4	PLL reference clock source, divider and output frequency	1079
33.2.5	MCG mode related APIs	1079
33.2.6	MCG auto trim machine(ATM) function	1079
33.2.7	Enumeration Type Documentation	1084
33.2.8	Function Documentation	1085

Contents

Section Number	Title	Page Number
Chapter	Multipurpose Clock Generator Lite (MCG_Lite)	
34.1	Overview	1113
34.2	MCG_Lite HAL driver	1114
34.2.1	Overview	1114
34.2.2	Get current clock output	1114
34.2.3	Clock mode switching	1114
34.2.4	Enumeration Type Documentation	1116
34.2.5	Function Documentation	1116
Chapter	Memory-Mapped Divide and Square Root(MMDVSQ)	
35.1	Overview	1129
35.2	MMDVSQ HAL driver	1130
35.2.1	Overview	1130
35.2.2	Perform MMDVSQ Divide or square root operation	1130
35.2.3	Function Documentation	1132
Chapter	Oscillator (OSC)	
36.1	Overview	1143
36.2	OSC HAL driver	1144
36.2.1	Overview	1144
36.2.2	Oscillator Control Register Access API Functions	1144
36.2.3	Enumeration Type Documentation	1145
36.2.4	Function Documentation	1145
36.3	Shared OSC Types	1148
Chapter	Power Management Controller (PMC)	
37.1	Overview	1149
37.2	OSC HAL driver	1150
37.2.1	Overview	1150
37.2.2	Power Management Controller feature access API functions	1150
37.2.3	Enumeration Type Documentation	1151
37.2.4	Function Documentation	1152
Chapter	Port Control and Interrupts (PORT)	
38.1	Overview	1159

Contents

Section Number	Title	Page Number
38.2	PORT HAL driver	.1160
38.2.1	Overview	.1160
38.2.2	Enumeration Type Documentation	.1161
38.2.3	Function Documentation	.1162
Chapter Reset Control Module (RCM)		
39.1	Overview	.1167
39.2	RCM HAL driver	.1168
39.2.1	Overview	.1168
39.2.2	Example	.1168
39.2.3	Function Documentation	.1169
Chapter System Integration Module (SIM)		
40.1	Overview	.1173
40.2	SIM HAL driver	.1174
40.2.1	Overview	.1174
40.2.2	Clock Gate Control register access APIs	.1174
40.2.3	Clock Source Control access APIs	.1174
40.2.4	Clock Divider access APIs	.1175
40.2.5	Enumeration Type Documentation	.1187
40.2.6	Function Documentation	.1188
40.2.7	K02F12810 SIM HAL driver	.1271
40.2.8	K22F12810 SIM HAL driver	.1278
40.2.9	K22F25612 SIM HAL driver	.1288
40.2.10	K22F51212 SIM HAL driver	.1298
40.2.11	KL03Z4 SIM HAL driver	.1308
40.2.12	K24F12 SIM HAL driver	.1314
40.2.13	KL46Z4 SIM HAL driver	.1323
40.2.14	K24F25612 SIM HAL driver	.1330
40.2.15	KV10Z7 SIM HAL driver	.1339
40.2.16	K60D10 SIM HAL driver	.1345
40.2.17	KV30F12810 SIM HAL driver	.1355
40.2.18	KV31F12810 SIM HAL driver	.1361
40.2.19	KV31F25612 SIM HAL driver	.1368
40.2.20	KV31F51212 SIM HAL driver	.1375
40.2.21	K64F12 SIM HAL driver	.1412
Chapter System Mode Controller (SMC)		
41.1	Overview	.1423

Contents

Section Number	Title	Page Number
41.2	SMC HAL driver	1424
41.2.1	Overview	1424
41.2.2	Power Mode Configuration APIs	1424
41.2.3	Data Structure Documentation	1427
41.2.4	Enumeration Type Documentation	1427
41.2.5	Function Documentation	1429
Chapter	Clock Manager (Clock)	
42.1	Overview	1435
42.1.1	Clock Manager	1446
42.2	Data Structure Documentation	1450
42.2.1	struct mcg_config_t	1450
42.2.2	struct clock_manager_user_config_t	1450
42.2.3	struct clock_notify_struct_t	1451
42.2.4	struct clock_manager_callback_user_config_t	1451
42.2.5	struct clock_manager_state_t	1452
42.3	Macro Definition Documentation	1452
42.3.1	CPU_LPO_CLK_HZ	1452
42.4	Typedef Documentation	1452
42.4.1	clock_manager_callback_t	1452
42.5	Enumeration Type Documentation	1453
42.5.1	clock_names_t	1453
42.5.2	clock_manager_error_code_t	1453
42.5.3	clock_manager_notify_t	1453
42.5.4	clock_manager_callback_type_t	1454
42.5.5	clock_manager_policy_t	1454
42.6	Function Documentation	1454
42.6.1	CLOCK_SYS_GetFreq	1454
42.6.2	CLOCK_SYS_GetCoreClockFreq	1454
42.6.3	CLOCK_SYS_GetSystemClockFreq	1455
42.6.4	CLOCK_SYS_GetBusClockFreq	1455
42.6.5	CLOCK_SYS_GetFlashClockFreq	1455
42.6.6	CLOCK_SYS_GetLpoClockFreq	1455
42.6.7	CLOCK_SYS_Init	1456
42.6.8	CLOCK_SYS_UpdateConfiguration	1457
42.6.9	CLOCK_SYS_SetConfiguration	1457
42.6.10	CLOCK_SYS_GetCurrentConfiguration	1458
42.6.11	CLOCK_SYS_GetErrorCallback	1458
42.6.12	CLOCK_SYS_SetMcgMode	1458

Contents

Section Number	Title	Page Number
42.6.13	CLOCK_SYS_SetOutDiv1	1459
42.6.14	CLOCK_SYS_GetOutDiv1	1459
42.6.15	CLOCK_SYS_SetOutDiv2	1459
42.6.16	CLOCK_SYS_GetOutDiv2	1460
42.6.17	CLOCK_SYS_SetOutDiv4	1460
42.6.18	CLOCK_SYS_GetOutDiv4	1460
42.6.19	CLOCK_SYS_SetOutDiv	1460
42.6.20	CLOCK_SYS_GetOutDiv	1461
42.6.21	CLOCK_SYS_GetPllfliClockFreq	1461
42.6.22	CLOCK_SYS_SetPllfliSel	1461
42.6.23	CLOCK_SYS_GetPllfliSel	1462
42.6.24	CLOCK_SYS_GetFixedFreqClockFreq	1462
42.6.25	CLOCK_SYS_GetInternalRefClockFreq	1462
42.6.26	CLOCK_SYS_GetExternalRefClock32kFreq	1462
42.6.27	CLOCK_SYS_SetExternalRefClock32kSrc	1463
42.6.28	CLOCK_SYS_GetExternalRefClock32kSrc	1464
42.6.29	CLOCK_SYS_GetOsc0ExternalRefClockFreq	1464
42.6.30	CLOCK_SYS_GetOsc0ExternalRefClockUndivFreq	1464
42.6.31	CLOCK_SYS_GetWdogFreq	1464
42.6.32	CLOCK_SYS_GetTraceSrc	1465
42.6.33	CLOCK_SYS_SetTraceSrc	1465
42.6.34	CLOCK_SYS_GetTraceFreq	1465
42.6.35	CLOCK_SYS_GetPortFilterFreq	1466
42.6.36	CLOCK_SYS_GetLptmrFreq	1466
42.6.37	CLOCK_SYS_GetEwmFreq	1467
42.6.38	CLOCK_SYS_GetFtfFreq	1467
42.6.39	CLOCK_SYS_GetCrcFreq	1468
42.6.40	CLOCK_SYS_GetCmpFreq	1469
42.6.41	CLOCK_SYS_GetVrefFreq	1470
42.6.42	CLOCK_SYS_GetPdbFreq	1470
42.6.43	CLOCK_SYS_GetPitFreq	1471
42.6.44	CLOCK_SYS_GetSpiFreq	1471
42.6.45	CLOCK_SYS_GetI2cFreq	1472
42.6.46	CLOCK_SYS_GetAdcAltFreq	1472
42.6.47	CLOCK_SYS_GetAdcAlt2Freq	1473
42.6.48	CLOCK_SYS_GetFtmFixedFreq	1473
42.6.49	CLOCK_SYS_GetFtmExternalFreq	1473
42.6.50	CLOCK_SYS_GetFtmExternalSrc	1474
42.6.51	CLOCK_SYS_SetFtmExternalSrc	1474
42.6.52	CLOCK_SYS_GetUartFreq	1474
42.6.53	CLOCK_SYS_GetGpioFreq	1475
42.6.54	CLOCK_SYS_EnableDmaClock	1476
42.6.55	CLOCK_SYS_DisableDmaClock	1476
42.6.56	CLOCK_SYS_GetDmaGateCmd	1476
42.6.57	CLOCK_SYS_EnableDmamuxClock	1477

Contents

Section Number	Title	Page Number
42.6.58	CLOCK_SYS_DisableDmamuxClock	1478
42.6.59	CLOCK_SYS_GetDmamuxGateCmd	1478
42.6.60	CLOCK_SYS_EnablePortClock	1478
42.6.61	CLOCK_SYS_DisablePortClock	1478
42.6.62	CLOCK_SYS_GetPortGateCmd	1479
42.6.63	CLOCK_SYS_EnableEwmClock	1479
42.6.64	CLOCK_SYS_DisableEwmClock	1479
42.6.65	CLOCK_SYS_GetEwmGateCmd	1480
42.6.66	CLOCK_SYS_EnableFtfClock	1480
42.6.67	CLOCK_SYS_DisableFtfClock	1480
42.6.68	CLOCK_SYS_GetFtfGateCmd	1480
42.6.69	CLOCK_SYS_EnableCrcClock	1481
42.6.70	CLOCK_SYS_DisableCrcClock	1481
42.6.71	CLOCK_SYS_GetCrcGateCmd	1481
42.6.72	CLOCK_SYS_EnableAdcClock	1482
42.6.73	CLOCK_SYS_DisableAdcClock	1483
42.6.74	CLOCK_SYS_GetAdcGateCmd	1483
42.6.75	CLOCK_SYS_EnableCmpClock	1483
42.6.76	CLOCK_SYS_DisableCmpClock	1484
42.6.77	CLOCK_SYS_GetCmpGateCmd	1484
42.6.78	CLOCK_SYS_EnableDacClock	1484
42.6.79	CLOCK_SYS_DisableDacClock	1484
42.6.80	CLOCK_SYS_GetDacGateCmd	1485
42.6.81	CLOCK_SYS_EnableVrefClock	1485
42.6.82	CLOCK_SYS_DisableVrefClock	1485
42.6.83	CLOCK_SYS_GetVrefGateCmd	1486
42.6.84	CLOCK_SYS_EnablePdbClock	1486
42.6.85	CLOCK_SYS_DisablePdbClock	1486
42.6.86	CLOCK_SYS_GetPdbGateCmd	1486
42.6.87	CLOCK_SYS_EnableFtmClock	1487
42.6.88	CLOCK_SYS_DisableFtmClock	1487
42.6.89	CLOCK_SYS_GetFtmGateCmd	1487
42.6.90	CLOCK_SYS_EnablePitClock	1487
42.6.91	CLOCK_SYS_DisablePitClock	1488
42.6.92	CLOCK_SYS_GetPitGateCmd	1488
42.6.93	CLOCK_SYS_EnableLptimerClock	1488
42.6.94	CLOCK_SYS_DisableLptimerClock	1488
42.6.95	CLOCK_SYS_GetLptimerGateCmd	1489
42.6.96	CLOCK_SYS_EnableSpiClock	1489
42.6.97	CLOCK_SYS_DisableSpiClock	1489
42.6.98	CLOCK_SYS_GetSpiGateCmd	1489
42.6.99	CLOCK_SYS_EnableI2cClock	1490
42.6.100	CLOCK_SYS_DisableI2cClock	1490
42.6.101	CLOCK_SYS_GetI2cGateCmd	1490
42.6.102	CLOCK_SYS_EnableUartClock	1491

Contents

Section Number	Title	Page Number
42.6.103	CLOCK_SYS_DisableUartClock	1491
42.6.104	CLOCK_SYS_GetUartGateCmd	1491
42.6.105	CLOCK_SYS_Osc0Init	1492
42.6.106	CLOCK_SYS_Osc0Deinit	1492
42.6.107	CLOCK_SYS_SetFtmExternalFreq	1492
42.6.108	CLOCK_SYS_GetRtcOutFreq	1493
42.6.109	CLOCK_SYS_GetRtcOutSrc	1493
42.6.110	CLOCK_SYS_SetRtcOutSrc	1493
42.6.111	CLOCK_SYS_GetCmtFreq	1493
42.6.112	CLOCK_SYS_GetUsbfsSrc	1494
42.6.113	CLOCK_SYS_SetUsbfsSrc	1494
42.6.114	CLOCK_SYS_GetUsbfsFreq	1495
42.6.115	CLOCK_SYS_SetUsbfsDiv	1495
42.6.116	CLOCK_SYS_GetUsbfsDiv	1495
42.6.117	CLOCK_SYS_GetLpuartSrc	1496
42.6.118	CLOCK_SYS_SetLpuartSrc	1496
42.6.119	CLOCK_SYS_GetLpuartFreq	1496
42.6.120	CLOCK_SYS_GetSaiFreq	1497
42.6.121	CLOCK_SYS_EnableSaiClock	1498
42.6.122	CLOCK_SYS_DisableSaiClock	1498
42.6.123	CLOCK_SYS_GetSaiGateCmd	1498
42.6.124	CLOCK_SYS_EnableRtcClock	1499
42.6.125	CLOCK_SYS_DisableRtcClock	1500
42.6.126	CLOCK_SYS_GetRtcGateCmd	1500
42.6.127	CLOCK_SYS_EnableUsbfsClock	1500
42.6.128	CLOCK_SYS_DisableUsbfsClock	1500
42.6.129	CLOCK_SYS_GetUsbfsGateCmd	1501
42.6.130	CLOCK_SYS_EnableLpuartClock	1501
42.6.131	CLOCK_SYS_DisableLpuartClock	1501
42.6.132	CLOCK_SYS_GetLpuartGateCmd	1502
42.6.133	CLOCK_SYS_SetUsbExternalFreq	1502
42.6.134	CLOCK_SYS_EnableRngaClock	1503
42.6.135	CLOCK_SYS_DisableRngaClock	1503
42.6.136	CLOCK_SYS_GetRngaGateCmd	1503
42.6.137	CLOCK_SYS_SetOutDiv3	1503
42.6.138	CLOCK_SYS_GetOutDiv3	1504
42.6.139	CLOCK_SYS_GetFlexbusFreq	1504
42.6.140	CLOCK_SYS_EnableFlexbusClock	1504
42.6.141	CLOCK_SYS_DisableFlexbusClock	1504
42.6.142	CLOCK_SYS_GetFlexbusGateCmd	1505
42.6.143	CLOCK_SYS_GetFlexcanFreq	1505
42.6.144	CLOCK_SYS_GetSdhcFreq	1505
42.6.145	CLOCK_SYS_SetSdhcSrc	1506
42.6.146	CLOCK_SYS_GetSdhcSrc	1506
42.6.147	CLOCK_SYS_GetUsbdcdFreq	1507

Contents

Section Number	Title	Page Number
42.6.148	CLOCK_SYS_EnableMpuClock	1507
42.6.149	CLOCK_SYS_DisableMpuClock	1508
42.6.150	CLOCK_SYS_GetMpuGateCmd	1509
42.6.151	CLOCK_SYS_EnableCmtClock	1509
42.6.152	CLOCK_SYS_DisableCmtClock	1509
42.6.153	CLOCK_SYS_GetCmtGateCmd	1509
42.6.154	CLOCK_SYS_EnableUsbdcdClock	1510
42.6.155	CLOCK_SYS_DisableUsbdcdClock	1510
42.6.156	CLOCK_SYS_GetUsbdcdGateCmd	1510
42.6.157	CLOCK_SYS_EnableFlexcanClock	1511
42.6.158	CLOCK_SYS_DisableFlexcanClock	1512
42.6.159	CLOCK_SYS_GetFlexcanGateCmd	1512
42.6.160	CLOCK_SYS_EnableSdhcClock	1512
42.6.161	CLOCK_SYS_DisableSdhcClock	1512
42.6.162	CLOCK_SYS_GetSdhcGateCmd	1513
42.6.163	CLOCK_SYS_SetSdhcExternalFreq	1513
42.6.164	CLOCK_SYS_GetEnetRmiiSrc	1513
42.6.165	CLOCK_SYS_SetEnetRmiiSrc	1514
42.6.166	CLOCK_SYS_GetEnetRmiiFreq	1515
42.6.167	CLOCK_SYS_SetEnetTimeStampSrc	1515
42.6.168	CLOCK_SYS_GetEnetTimeStampSrc	1516
42.6.169	CLOCK_SYS_GetEnetTimeStampFreq	1516
42.6.170	CLOCK_SYS_EnableEnetClock	1517
42.6.171	CLOCK_SYS_DisableEnetClock	1517
42.6.172	CLOCK_SYS_GetEnetGateCmd	1517
42.6.173	CLOCK_SYS_EnableTsiClock	1518
42.6.174	CLOCK_SYS_DisableTsiClock	1519
42.6.175	CLOCK_SYS_GetTsiGateCmd	1519
42.6.176	CLOCK_SYS_EnableLlwuClock	1519
42.6.177	CLOCK_SYS_DisableLlwuClock	1519
42.6.178	CLOCK_SYS_GetLlwuGateCmd	1520
42.6.179	CLOCK_SYS_SetEnetExternalFreq	1520
42.6.180	CLOCK_SYS_GetDmaFreq	1520
42.6.181	CLOCK_SYS_GetDmamuxFreq	1521
42.6.182	CLOCK_SYS_GetPortFreq	1521
42.6.183	CLOCK_SYS_GetMpuFreq	1522
42.6.184	CLOCK_SYS_GetFlexbusFreq	1522
42.6.185	CLOCK_SYS_GetRngaFreq	1523
42.6.186	CLOCK_SYS_GetAdcFreq	1523
42.6.187	CLOCK_SYS_GetFtmFreq	1524
42.6.188	CLOCK_SYS_GetUsbFreq	1524
42.6.189	CLOCK_SYS_GetSaiFreq	1525
42.6.190	CLOCK_SYS_EnableUsbClock	1525
42.6.191	CLOCK_SYS_DisableUsbClock	1526
42.6.192	CLOCK_SYS_GetUsbGateCmd	1526

Contents

Section Number	Title	Page Number
42.6.193	CLOCK_SYS_GetTpmFreq	1526
42.6.194	CLOCK_SYS_SetTpmSrc	1527
42.6.195	CLOCK_SYS_GetTpmSrc	1527
42.6.196	CLOCK_SYS_GetTpmExternalFreq	1527
42.6.197	CLOCK_SYS_SetTpmExternalSrc	1528
42.6.198	CLOCK_SYS_GetTpmExternalSrc	1528
42.6.199	CLOCK_SYS_EnableTpmClock	1528
42.6.200	CLOCK_SYS_DisableTpmClock	1529
42.6.201	CLOCK_SYS_GetTpmGateCmd	1529
42.6.202	CLOCK_SYS_SetTpmExternalFreq	1529
42.6.203	CLOCK_SYS_GetCopFreq	1530
42.6.204	CLOCK_SYS_GetRtcFreq	1530
42.6.205	CLOCK_SYS_GetLpscifreq	1530
42.6.206	CLOCK_SYS_SetLpscifsrc	1531
42.6.207	CLOCK_SYS_GetLpscifsrc	1531
42.6.208	CLOCK_SYS_EnableLpsciclock	1531
42.6.209	CLOCK_SYS_DisableLpsciclock	1532
42.6.210	CLOCK_SYS_GetLpscigatecmd	1532
42.6.211	CLOCK_SYS_SetOutDiv5	1532
42.6.212	CLOCK_SYS_GetOutDiv5	1532
42.6.213	CLOCK_SYS_GetOutdiv5ClockFreq	1533
42.7	Variable Documentation	1533
42.7.1	g_simBaseAddr	1533
42.7.2	g_mcgBaseAddr	1533
42.7.3	g_oscBaseAddr	1533
42.7.4	g_defaultClockConfigurations	1533
42.8	K02F12810	1534
42.8.1	Overview	1534
42.8.2	Data Structure Documentation	1534
42.8.3	Macro Definition Documentation	1536
42.8.4	Variable Documentation	1536
42.9	K22F12810	1537
42.9.1	Overview	1537
42.9.2	Data Structure Documentation	1538
42.9.3	Macro Definition Documentation	1539
42.9.4	Variable Documentation	1539
42.10	K22F25612	1540
42.10.1	Overview	1540
42.10.2	Data Structure Documentation	1541
42.10.3	Macro Definition Documentation	1542
42.10.4	Variable Documentation	1542

Contents

Section Number	Title	Page Number
42.11	K22F51212	1543
42.11.1	Overview	1543
42.11.2	Data Structure Documentation	1544
42.11.3	Macro Definition Documentation	1545
42.11.4	Variable Documentation	1545
42.12	K24F12	1546
42.12.1	Overview	1546
42.12.2	Data Structure Documentation	1547
42.12.3	Macro Definition Documentation	1548
42.12.4	Variable Documentation	1548
42.13	K24F25612	1549
42.13.1	Overview	1549
42.13.2	Data Structure Documentation	1549
42.13.3	Macro Definition Documentation	1551
42.13.4	Variable Documentation	1551
42.14	K60D10	1552
42.14.1	Overview	1552
42.14.2	Data Structure Documentation	1553
42.14.3	Macro Definition Documentation	1555
42.14.4	Variable Documentation	1555
42.15	K64F12	1556
42.15.1	Overview	1556
42.15.2	Data Structure Documentation	1557
42.15.3	Macro Definition Documentation	1559
42.15.4	Variable Documentation	1559
42.16	KL03Z4	1560
42.16.1	Overview	1560
42.16.2	Data Structure Documentation	1560
42.16.3	Macro Definition Documentation	1561
42.16.4	Variable Documentation	1561
42.17	KL46Z4	1562
42.17.1	Overview	1562
42.17.2	Data Structure Documentation	1562
42.17.3	Macro Definition Documentation	1564
42.17.4	Variable Documentation	1564
42.18	KV10Z7	1565
42.18.1	Overview	1565
42.18.2	Data Structure Documentation	1565
42.18.3	Macro Definition Documentation	1567

Contents

Section Number	Title	Page Number
42.18.4	Variable Documentation	1567
42.19	KV30F12810	1568
42.19.1	Overview	1568
42.19.2	Data Structure Documentation	1568
42.19.3	Macro Definition Documentation	1570
42.19.4	Variable Documentation	1570
42.20	KV31F12810	1571
42.20.1	Overview	1571
42.20.2	Data Structure Documentation	1571
42.20.3	Macro Definition Documentation	1573
42.20.4	Variable Documentation	1573
42.21	KV31F25612	1574
42.21.1	Overview	1574
42.21.2	Data Structure Documentation	1574
42.21.3	Macro Definition Documentation	1576
42.21.4	Variable Documentation	1576
42.22	KV31F51212	1577
42.22.1	Overview	1577
42.22.2	Data Structure Documentation	1577
42.22.3	Macro Definition Documentation	1579
42.22.4	Variable Documentation	1579
 Chapter Hwtimer_driver		
43.1	Overview	1581
43.2	HwTimer Driver	1582
43.2.1	HwTimer Overview	1582
 Chapter Interrupt Manager (Interrupt)		
44.1	Overview	1583
44.1.1	Interrupt Manager	1583
44.2	Enumeration Type Documentation	1584
44.2.1	interrupt_status_t	1584
44.3	Function Documentation	1584
44.3.1	INT_SYS_InstallHandler	1584
44.3.2	INT_SYS_EnableIRQ	1584
44.3.3	INT_SYS_DisableIRQ	1585
44.3.4	INT_SYS_EnableIRQGlobal	1585

Contents

Section Number	Title	Page Number
44.3.5	INT_SYS_DisableIRQGlobal	1585
Chapter Power Manager (Power)		
45.1	Overview	1587
45.2	Data Structure Documentation	1589
45.2.1	struct power_manager_user_config_t	1589
45.2.2	struct power_manager_notify_struct_t	1590
45.2.3	struct power_manager_callback_user_config_t	1590
45.2.4	struct power_manager_state_t	1590
45.3	Typedef Documentation	1591
45.3.1	power_manager_callback_data_t	1591
45.3.2	power_manager_callback_t	1591
45.4	Enumeration Type Documentation	1592
45.4.1	power_manager_modes_t	1592
45.4.2	power_manager_error_code_t	1593
45.4.3	power_manager_policy_t	1593
45.4.4	power_manager_notify_t	1593
45.4.5	power_manager_callback_type_t	1594
45.5	Function Documentation	1594
45.5.1	POWER_SYS_Init	1594
45.5.2	POWER_SYS_Deinit	1595
45.5.3	POWER_SYS_SetMode	1596
45.5.4	POWER_SYS_GetLastMode	1596
45.5.5	POWER_SYS_GetLastModeConfig	1597
45.5.6	POWER_SYS_GetCurrentMode	1597
45.5.7	POWER_SYS_GetErrorCallbackIndex	1597
45.5.8	POWER_SYS_GetErrorCallback	1598
45.5.9	POWER_SYS_GetVeryLowPowerModeStatus	1598
45.5.10	POWER_SYS_GetLowLeakageWakeupResetStatus	1598
45.5.11	POWER_SYS_GetAckIsolation	1598
45.5.12	POWER_SYS_ClearAckIsolation	1599
45.6	Power Manager driver	1600
45.6.1	Power Manager Overview	1600
Chapter Utilities for the Kinetis SDK		
46.1	Overview	1603
46.2	Debug_console	1604

Contents

Section Number	Title	Page Number
Chapter	OS Abstraction Layer (OSA)	
47.1	Overview	1605
47.1.1	OS Abstraction Layer	1607
47.2	Enumeration Type Documentation	1613
47.2.1	osa_status_t	1613
47.2.2	osa_event_clear_mode_t	1613
47.2.3	osa_critical_section_mode_t	1613
47.3	Function Documentation	1614
47.3.1	OSA_SemaCreate	1614
47.3.2	OSA_SemaWait	1615
47.3.3	OSA_SemaPost	1616
47.3.4	OSA_SemaDestroy	1616
47.3.5	OSA_MutexCreate	1617
47.3.6	OSA_MutexLock	1618
47.3.7	OSA_MutexUnlock	1619
47.3.8	OSA_MutexDestroy	1619
47.3.9	OSA_EventCreate	1620
47.3.10	OSA_EventWait	1620
47.3.11	OSA_EventSet	1621
47.3.12	OSA_EventClear	1622
47.3.13	OSA_EventGetFlags	1623
47.3.14	OSA_EventDestroy	1623
47.3.15	OSA_TaskCreate	1624
47.3.16	OSA_TaskDestroy	1625
47.3.17	OSA_TaskYield	1625
47.3.18	OSA_TaskGetHandler	1626
47.3.19	OSA_TaskGetPriority	1626
47.3.20	OSA_TaskSetPriority	1626
47.3.21	OSA_MsgQCreate	1627
47.3.22	OSA_MsgQPut	1627
47.3.23	OSA_MsgQGet	1628
47.3.24	OSA_MsgQDestroy	1629
47.3.25	OSA_MemAlloc	1629
47.3.26	OSA_MemAllocZero	1629
47.3.27	OSA_MemFree	1630
47.3.28	OSA_TimeDelay	1630
47.3.29	OSA_TimeGetMsec	1630
47.3.30	OSA_InstallIntHandler	1630
47.3.31	OSA_EnterCritical	1631
47.3.32	OSA_ExitCritical	1631
47.3.33	OSA_Init	1631
47.3.34	OSA_Start	1632

Contents

Section Number	Title	Page Number
47.4	Bare Metal Abstraction Layer	1634
47.4.1	Overview	1634
47.4.2	Data Structure Documentation	1637
47.4.3	Macro Definition Documentation	1639
47.4.4	Function Documentation	1640
47.5	MQX Abstraction Layer	1641
47.5.1	Overview	1641
47.5.2	Data Structure Documentation	1642
47.5.3	Macro Definition Documentation	1643
47.5.4	Typedef Documentation	1644
47.6	μC/OS-II Abstraction Layer	1645
47.6.1	Overview	1645
47.6.2	Data Structure Documentation	1646
47.6.3	Macro Definition Documentation	1647
47.7	μC/OS-III Abstraction Layer	1648
47.7.1	Overview	1648
47.7.2	Data Structure Documentation	1649
47.7.3	Macro Definition Documentation	1650
47.7.4	Typedef Documentation	1651
47.8	FreeRTOS Abstraction Layer	1652
47.8.1	Overview	1652
47.8.2	Data Structure Documentation	1653
47.8.3	Macro Definition Documentation	1653
47.8.4	Typedef Documentation	1654
Chapter	Cop_driver	
48.1	Overview	1655
48.2	Data Structure Documentation	1655
48.2.1	struct cop_user_config_t	1655
48.3	Function Documentation	1656
48.3.1	COP_DRV_Init	1656
48.3.2	COP_DRV_Deinit	1656
48.3.3	COP_DRV_Refresh	1656
48.3.4	COP_DRV_IsRunning	1656
48.3.5	COP_DRV_ResetSystem	1657
48.4	Variable Documentation	1657
48.4.1	g_copBaseAddr	1657

Contents

Section Number	Title	Page Number
Chapter Crc_driver		
49.1	Overview	1659
49.2	Data Structure Documentation	1659
49.2.1	struct crc_user_config_t	1659
49.3	Function Documentation	1660
49.3.1	CRC_DRV_Init	1660
49.3.2	CRC_DRV_Deinit	1660
49.3.3	CRC_DRV_GetCrcBlock	1660
49.3.4	CRC_DRV_Configure	1661
49.4	Variable Documentation	1661
49.4.1	g_crcBaseAddr	1661
Chapter Enet_driver		
50.1	Overview	1663
50.2	Data Structure Documentation	1665
50.2.1	struct enet_multicast_group_t	1665
50.2.2	struct enet_ethernet_header_t	1665
50.2.3	struct enet_8021vlan_header_t	1665
50.2.4	struct enet_buff_descrip_context_t	1666
50.2.5	struct enet_stats_t	1666
50.2.6	struct enet_mac_packet_buffer_t	1667
50.2.7	struct enet_buff_config_t	1667
50.2.8	struct enet_dev_if_t	1667
50.3	Macro Definition Documentation	1668
50.3.1	ENET_ENABLE_DETAIL_STATS	1668
50.3.2	ENET_ORIGINAL_CRC32	1668
50.4	Enumeration Type Documentation	1668
50.4.1	enet_crc_parameter_t	1668
50.4.2	enet_protocol_type_t	1668
50.5	Function Documentation	1669
50.5.1	ENET_DRV_Init	1669
50.5.2	ENET_DRV_Deinit	1669
50.5.3	ENET_DRV_UpdateRxBuffDescrip	1669
50.5.4	ENET_DRV_CleanupTxBuffDescrip	1670
50.5.5	ENET_DRV_IncrRxBuffDescripIndex	1670
50.5.6	ENET_DRV_IncrTxBuffDescripIndex	1670
50.5.7	ENET_DRV_RxErrorStats	1671

Contents

Section Number	Title	Page Number
50.5.8	ENET_DRV_TxErrorStats	1671
50.5.9	ENET_DRV_ReceiveData	1671
50.5.10	ENET_DRV_InstallNetIfCall	1672
50.5.11	ENET_DRV_SendData	1672
50.5.12	ENET_DRV_RxIRQHandler	1672
50.5.13	ENET_DRV_TxIRQHandler	1673
50.5.14	ENET_DRV_CalculateCrc32	1673
50.5.15	ENET_DRV_AddMulticastGroup	1673
50.5.16	ENET_DRV_LeaveMulticastGroup	1673
50.5.17	enet_mac_enqueue_buffer	1674
50.5.18	enet_mac_dequeue_buffer	1674
50.6	Variable Documentation	1674
50.6.1	g_enetBaseAddr	1674
50.6.2	g_enetTxIrqId	1674
Chapter Ewm_driver		
51.1	Overview	1675
51.2	Data Structure Documentation	1675
51.2.1	struct ewm_user_config_t	1675
51.3	Function Documentation	1676
51.3.1	EWM_DRV_Init	1676
51.3.2	EWM_DRV_Deinit	1676
51.3.3	EWM_DRV_Refresh	1676
51.3.4	EWM_DRV_IsRunning	1676
51.3.5	EWM_DRV_GetIntCmd	1677
51.3.6	EWM_DRV_SetIntCmd	1677
51.4	Variable Documentation	1677
51.4.1	g_ewmBaseAddr	1677
51.4.2	g_ewmIrqId	1677
Chapter Lpisci_driver		
52.1	Overview	1679
52.2	Data Structure Documentation	1680
52.2.1	struct lpisci_state_t	1680
52.2.2	struct lpisci_user_config_t	1681
52.3	Typedef Documentation	1681
52.3.1	lpisci_rx_callback_t	1681

Contents

Section Number	Title	Page Number
52.4	Function Documentation	1682
52.4.1	LPSCI_DRV_Init	1682
52.4.2	LPSCI_DRV_Deinit	1682
52.4.3	LPSCI_DRV_InstallRxCallback	1683
52.4.4	LPSCI_DRV_SendDataBlocking	1683
52.4.5	LPSCI_DRV_SendData	1684
52.4.6	LPSCI_DRV_GetTransmitStatus	1684
52.4.7	LPSCI_DRV_AbortSendingData	1685
52.4.8	LPSCI_DRV_ReceiveDataBlocking	1686
52.4.9	LPSCI_DRV_ReceiveData	1686
52.4.10	LPSCI_DRV_GetReceiveStatus	1687
52.4.11	LPSCI_DRV_AbortReceivingData	1687
52.5	Variable Documentation	1688
52.5.1	g_lpisciBaseAddr	1688
Chapter	Dspi_driver_unit_test	
53.1	Overview	1689
53.2	Function Documentation	1689
53.2.1	dspi_sameboard_loopback_test	1689
53.2.2	dspi_edma_self_loopback_test_vary_transfer_cnt	1689
53.2.3	dspi_edma_self_loopback_test_vary_bitcnt_baudrate	1689
Chapter	Enet_rtcs_adaptor	
54.1	Overview	1691
54.2	Data Structure Documentation	1692
54.2.1	struct ENET_HEADER	1692
54.2.2	struct ENET_8021QTAG_HEADER	1693
54.2.3	struct ENET_8022_HEADER	1693
54.2.4	struct pcb_queue	1693
54.2.5	struct enet_ecb_struct_t	1694
54.2.6	struct ENET_COMMON_STATS_STRUCT	1694
54.3	Function Documentation	1694
54.3.1	ENET_initialize	1694
54.3.2	ENET_open	1694
54.3.3	ENET_shutdown	1695
54.3.4	ENET_receive	1695
54.3.5	ENET_send	1695
54.3.6	ENET_get_address	1696
54.3.7	ENET_get_mac_address	1696

Contents

Section Number	Title	Page Number
54.3.8	ENET_join	1696
54.3.9	ENET_leave	1697
54.3.10	ENET_link_status	1697
54.3.11	ENET_get_speed	1697
54.3.12	ENET_get_MTU	1698
54.3.13	ENET_phy_registers	1698
54.3.14	ENET_get_options	1698
54.3.15	ENET_close	1699
54.3.16	ENET_mediactl	1699
54.3.17	ENET_get_next_device_handle	1699
54.3.18	ENET_free	1700
54.3.19	ENET_strerror	1700
Chapter	Ewm_hal_unit_test	
55.1	Overview	1701
Chapter	Lpisci_driver_unit_test	
56.1	Overview	1703
56.2	Function Documentation	1703
56.2.1	lpisci_send_and_abort_test	1703
56.2.2	lpisci_send_receive_driver_test	1703
56.2.3	lpisci_async_driver_test	1703
Chapter	LPTMR_HAL_unit_test	
57.1	Overview	1705
57.2	Data Structure Documentation	1705
57.2.1	union lptmr_test_status_t	1705
Chapter	Lpuart_driver_unit_test	
58.1	Overview	1707
58.2	Function Documentation	1707
58.2.1	LPUART_DRV_SendAndAbortTest	1707
58.2.2	LPUART_DRV_SendReceiveDriverTest	1707
58.2.3	LPUART_DRV_AsyncDriverTest	1707
Chapter	MPU_HAL_unit_test	
59.1	Overview	1709

Contents

Section Number	Title	Page Number
Chapter	Spi_driver_unit_test	
60.1	Overview	.1711
60.2	Function Documentation	.1711
60.2.1	spi_sameboard_loopback_test	.1711
60.2.2	spi_test_pinmux_setup	.1711
60.2.3	spi_dma_self_loopback_test_vary_transfer_cnt	.1711
Chapter	Uart_driver_unit_test	
61.1	Overview	.1713
61.2	Function Documentation	.1713
61.2.1	uart_send_and_abort_test	.1713
61.2.2	uart_send_receive_driver_test	.1713
61.2.3	uart_async_driver_test	.1713
Chapter	Wdog_hal_unit_test	
62.1	Overview	.1715
Chapter	Crc_hal	
63.1	Overview	.1717
63.2	Enumeration Type Documentation	.1719
63.2.1	crc_status_t	.1719
63.2.2	crc_transpose_t	.1719
63.2.3	crc_prot_width_t	.1719
63.3	Function Documentation	.1719
63.3.1	CRC_HAL_Init	.1719
63.3.2	CRC_HAL_GetDataReg	.1719
63.3.3	CRC_HAL_GetDataHReg	.1720
63.3.4	CRC_HAL_GetDataLReg	.1720
63.3.5	CRC_HAL_SetDataReg	.1720
63.3.6	CRC_HAL_SetDataHReg	.1721
63.3.7	CRC_HAL_SetDataLReg	.1721
63.3.8	CRC_HAL_SetDataHLReg	.1721
63.3.9	CRC_HAL_SetDataHUReg	.1721
63.3.10	CRC_HAL_SetDataLLReg	.1722
63.3.11	CRC_HAL_SetDataLUReg	.1722
63.3.12	CRC_HAL_GetPolyReg	.1722
63.3.13	CRC_HAL_GetPolyHReg	.1722

Contents

Section Number	Title	Page Number
63.3.14	CRC_HAL_GetPolyLReg	1723
63.3.15	CRC_HAL_SetPolyReg	1723
63.3.16	CRC_HAL_SetPolyHReg	1723
63.3.17	CRC_HAL_SetPolyLReg	1724
63.3.18	CRC_HAL_GetCtrlReg	1724
63.3.19	CRC_HAL_SetCtrlReg	1724
63.3.20	CRC_HAL_GetSeedOrDataMode	1724
63.3.21	CRC_HAL_SetSeedOrDataMode	1725
63.3.22	CRC_HAL_GetWriteTranspose	1725
63.3.23	CRC_HAL_SetWriteTranspose	1725
63.3.24	CRC_HAL_GetReadTranspose	1726
63.3.25	CRC_HAL_SetReadTranspose	1726
63.3.26	CRC_HAL_GetXorMode	1726
63.3.27	CRC_HAL_SetXorMode	1727
63.3.28	CRC_HAL_GetProtocolWidth	1728
63.3.29	CRC_HAL_SetProtocolWidth	1728
63.3.30	CRC_HAL_GetCrc32	1728
63.3.31	CRC_HAL_GetCrc16	1729
63.3.32	CRC_HAL_GetCrc8	1729
63.3.33	CRC_HAL_GetCrcResult	1730

Chapter [Flexbus_hal](#)

64.1	Overview	1731
64.2	Enumeration Type Documentation	1734
64.2.1	flexbus_status_t	1734
64.2.2	flexbus_port_size_t	1734
64.2.3	flexbus_write_address_hold_t	1734
64.2.4	flexbus_read_address_hold_t	1734
64.2.5	flexbus_address_setup_t	1734
64.2.6	flexbus_bytelane_shift_t	1734
64.2.7	flexbus_multiplex_group1_t	1734
64.2.8	flexbus_multiplex_group2_t	1734
64.2.9	flexbus_multiplex_group3_t	1734
64.2.10	flexbus_multiplex_group4_t	1734
64.2.11	flexbus_multiplex_group5_t	1734
64.3	Function Documentation	1734
64.3.1	FB_HAL_Init	1734
64.3.2	FB_HAL_WriteBaseAddr	1734
64.3.3	FB_HAL_ReadBaseAddr	1735
64.3.4	FB_HAL_SetChipSelectValidCmd	1735
64.3.5	FB_HAL_IsChipSelectValid	1736
64.3.6	FB_HAL_SetWriteProtectionCmd	1736

Contents

Section Number	Title	Page Number
64.3.7	FB_HAL_IsWriteProtectionEnabled	1736
64.3.8	FB_HAL_WriteBaseAddrMask	1737
64.3.9	FB_HAL_ReadBaseAddrMask	1737
64.3.10	FB_HAL_SetBurstWriteCmd	1738
64.3.11	FB_HAL_IsBurstWriteEnabled	1738
64.3.12	FB_HAL_SetBurstReadCmd	1738
64.3.13	FB_HAL_IsBurstReadEnabled	1740
64.3.14	FB_HAL_SetByteModeCmd	1740
64.3.15	FB_HAL_IsByteEnableModeEnabled	1741
64.3.16	FB_HAL_SetPortSize	1741
64.3.17	FB_HAL_GetPortSize	1741
64.3.18	FB_HAL_SetAutoAcknowledgeCmd	1742
64.3.19	FB_HAL_IsAutoAcknowledgeEnabled	1742
64.3.20	FB_HAL_SetByteLaneShift	1743
64.3.21	FB_HAL_GetByteLaneShift	1743
64.3.22	FB_HAL_SetWaitStates	1744
64.3.23	FB_HAL_GetWaitStates	1745
64.3.24	FB_HAL_SetWriteAddrHoldOrDeselect	1745
64.3.25	FB_HAL_GetWriteAddrHoldOrDeselect	1746
64.3.26	FB_HAL_SetReadAddrHoldOrDeselect	1746
64.3.27	FB_HAL_GetReadAddrHoldOrDeselect	1746
64.3.28	FB_HAL_SetAddrSetup	1747
64.3.29	FB_HAL_GetAddrSetup	1747
64.3.30	FB_HAL_SetExtendedAddrLatchCmd	1748
64.3.31	FB_HAL_IsExtendedAddrLatchEnabled	1748
64.3.32	FB_HAL_SetSecondaryWaitStateCmd	1748
64.3.33	FB_HAL_IsSecondaryWaitStateEnabled	1749
64.3.34	FB_HAL_SetMultiplexControlGroup1	1749
64.3.35	FB_HAL_GetMultiplexControlGroup1	1750
64.3.36	FB_HAL_SetMultiplexControlGroup2	1751
64.3.37	FB_HAL_GetMultiplexControlGroup2	1751
64.3.38	FB_HAL_SetMultiplexControlGroup3	1751
64.3.39	FB_HAL_GetMultiplexControlGroup3	1752
64.3.40	FB_HAL_SetMultiplexControlGroup4	1752
64.3.41	FB_HAL_GetMultiplexControlGroup4	1752
64.3.42	FB_HAL_SetMultiplexControlGroup5	1753
64.3.43	FB_HAL_GetMultiplexControlGroup5	1753

Chapter [Flexio_hal](#)

65.1	Overview	1755
65.2	Data Structure Documentation	1762
65.2.1	struct flexio_shifter_t	1762
65.2.2	struct flexio_timer_t	1763

Contents

Section Number	Title	Page Number
65.3	Enumeration Type Documentation	1764
65.3.1	flexio_shifter_pin_configure_t	1764
65.3.2	flexio_pin_polarity_t	1764
65.3.3	flexio_shifter_mode_t	1764
65.3.4	flexio_shifter_input_source_t	1764
65.3.5	flexio_shifter_stop_bit_t	1764
65.3.6	flexio_shifter_start_bit_t	1765
65.3.7	flexio_timer_trigger_polarity_t	1765
65.3.8	flexio_timer_trigger_source_t	1765
65.3.9	flexio_timer_pin_config_t	1765
65.3.10	flexio_timer_mode_t	1765
65.3.11	flexio_timer_output_t	1766
65.3.12	flexio_timer_decrement_source_t	1766
65.3.13	flexio_timer_reset_condition_t	1766
65.3.14	flexio_timer_enable_condition_t	1767
65.3.15	flexio_timer_disable_condition_t	1767
65.3.16	flexio_timer_stop_bit_condition_t	1767
65.4	Function Documentation	1767
65.4.1	FLEXIO_HAL_Init	1767
65.4.2	FLEXIO_HAL_SetModuleEnable	1768
65.4.3	FLEXIO_HAL_GetModuleEnable	1768
65.4.4	FLEXIO_HAL_SetDozeMode	1768
65.4.5	FLEXIO_HAL_GetDozeMode	1768
65.4.6	FLEXIO_HAL_SetDebugMode	1769
65.4.7	FLEXIO_HAL_GetDebugMode	1769
65.4.8	FLEXIO_HAL_SetFastAccess	1769
65.4.9	FLEXIO_HAL_GetFastAccess	1770
65.4.10	FLEXIO_HAL_SetSoftwareReset	1770
65.4.11	FLEXIO_HAL_GetSoftwareReset	1770
65.4.12	FLEXIO_HAL_GetShifterStatusFlag	1770
65.4.13	FLEXIO_HAL_ClearShifterStatusFlag	1771
65.4.14	FLEXIO_HAL_GetShifterErrorFlag	1771
65.4.15	FLEXIO_HAL_ClearShifterErrorFlag	1771
65.4.16	FLEXIO_HAL_GetTimerStatusFlag	1772
65.4.17	FLEXIO_HAL_ClearTimerStatusFlag	1772
65.4.18	FLEXIO_HAL_SetShifterStatusInt	1772
65.4.19	FLEXIO_HAL_GetShifterStatusInt	1773
65.4.20	FLEXIO_HAL_SetShifterErrorInt	1773
65.4.21	FLEXIO_HAL_GetShifterErrorInt	1773
65.4.22	FLEXIO_HAL_SetTimerStatusInt	1774
65.4.23	FLEXIO_HAL_GetTimerStatusInt	1775
65.4.24	FLEXIO_HAL_SetShifterStatusDma	1775
65.4.25	FLEXIO_HAL_GetShiftStatusDma	1775
65.4.26	FLEXIO_HAL_SetShifterTimer	1776

Contents

Section Number	Title	Page Number
65.4.27	FLEXIO_HAL_GetShifterTimer	1777
65.4.28	FLEXIO_HAL_SetTimerPolarity	1777
65.4.29	FLEXIO_HAL_GetTimerPolarity	1777
65.4.30	FLEXIO_HAL_SetShifterPinConfig	1778
65.4.31	FLEXIO_HAL_GetShifterPinConfig	1778
65.4.32	FLEXIO_HAL_SetShifterPin	1778
65.4.33	FLEXIO_HAL_GetShifterPin	1778
65.4.34	FLEXIO_HAL_SetShifterPinPolarity	1779
65.4.35	FLEXIO_HAL_GetShifterPinPolarity	1779
65.4.36	FLEXIO_HAL_SetShifterMode	1779
65.4.37	FLEXIO_HAL_GetShifterMode	1780
65.4.38	FLEXIO_HAL_SetShifterInputSource	1780
65.4.39	FLEXIO_HAL_GetShifterInputSource	1780
65.4.40	FLEXIO_HAL_SetShifterStopBit	1781
65.4.41	FLEXIO_HAL_GetShifterStopBit	1781
65.4.42	FLEXIO_HAL_SetShifterStartBit	1781
65.4.43	FLEXIO_HAL_GetShifterStartBit	1781
65.4.44	FLEXIO_HAL_GetShifterBuffer	1782
65.4.45	FLEXIO_HAL_GetShifterBufferBitByteSwapped	1782
65.4.46	FLEXIO_HAL_GetShifterBufferByteSwapped	1782
65.4.47	FLEXIO_HAL_GetShifterBufferBitSwapped	1783
65.4.48	FLEXIO_HAL_SetShifterBuffer	1783
65.4.49	FLEXIO_HAL_SetShifterBufferBitByteSwapped	1783
65.4.50	FLEXIO_HAL_SetShifterBufferByteSwapped	1784
65.4.51	FLEXIO_HAL_SetShifterBufferBitSwapped	1784
65.4.52	FLEXIO_HAL_SetTimerTrigger	1784
65.4.53	FLEXIO_HAL_GetTimerTrigger	1785
65.4.54	FLEXIO_HAL_SetTimerTriggerPolarity	1785
65.4.55	FLEXIO_HAL_GetTimerTriggerPolarity	1785
65.4.56	FLEXIO_HAL_SetTimerTriggerSource	1786
65.4.57	FLEXIO_HAL_GetTimerTriggerSource	1786
65.4.58	FLEXIO_HAL_SetTimerPinConfig	1786
65.4.59	FLEXIO_HAL_GetTimerPinConfig	1787
65.4.60	FLEXIO_HAL_SetTimerPin	1787
65.4.61	FLEXIO_HAL_GetTimerPin	1787
65.4.62	FLEXIO_HAL_SetTimerPinPolarity	1788
65.4.63	FLEXIO_HAL_GetTimerPinPolarity	1788
65.4.64	FLEXIO_HAL_SetTimerMode	1788
65.4.65	FLEXIO_HAL_GetTimerMode	1789
65.4.66	FLEXIO_HAL_SetTimerOutput	1789
65.4.67	FLEXIO_HAL_GetTimerOutput	1789
65.4.68	FLEXIO_HAL_SetTimerDecrementSource	1790
65.4.69	FLEXIO_HAL_GetTimerDecrementSource	1790
65.4.70	FLEXIO_HAL_SetTimerResetCondition	1790
65.4.71	FLEXIO_HAL_GetTimerResetCondition	1791

Contents

Section Number	Title	Page Number
65.4.72	FLEXIO_HAL_SetTimerEnableCondition	1791
65.4.73	FLEXIO_HAL_GetTimerEnableCondition	1791
65.4.74	FLEXIO_HAL_SetTimerDisableCondition	1792
65.4.75	FLEXIO_HAL_GetTimerDisableCondition	1792
65.4.76	FLEXIO_HAL_SetTimerStopBitCondition	1792
65.4.77	FLEXIO_HAL_GetTimerStopBitCondition	1793
65.4.78	FLEXIO_HAL_SetTimerStartBit	1793
65.4.79	FLEXIO_HAL_GetTimerStartBit	1793
65.4.80	FLEXIO_HAL_SetTimerCompareValue	1794
65.4.81	FLEXIO_HAL_GetTimerCompareValue	1794
65.4.82	FLEXIO_HAL_GetMajorVersionNumber	1794
65.4.83	FLEXIO_HAL_GetMinorVersionNumber	1795
65.4.84	FLEXIO_HAL_GetFeatureNumber	1795
65.4.85	FLEXIO_HAL_GetTriggerNumber	1795
65.4.86	FLEXIO_HAL_GetPinNumber	1796
65.4.87	FLEXIO_HAL_GetTimerNumber	1796
65.4.88	FLEXIO_HAL_GetShifterNumber	1796
65.4.89	FLEXIO_HAL_GetPinStatus	1797
65.4.90	FLEXIO_HAL_ConfigureTimer	1797
65.4.91	FLEXIO_HAL_ConfigureShifter	1797

Chapter [Flexio_uart_hal](#)

66.1	Overview	1799
66.2	Function Documentation	1799
66.2.1	FLEXIO_UART_HAL_Init	1799
66.2.2	FLEXIO_UART_HAL_GetChar	1800
66.2.3	FLEXIO_UART_HAL_PutChar	1800
66.2.4	FLEXIO_UART_HAL_IsRxReady	1800
66.2.5	FLEXIO_UART_HAL_IsTxReady	1801
66.2.6	FLEXIO_UART_HAL_Deinit	1801

Chapter [Lpisci_hal](#)

67.1	Overview	1803
67.2	Enumeration Type Documentation	1807
67.2.1	lpisci_status_t	1807
67.2.2	lpisci_stop_bit_count_t	1807
67.2.3	lpisci_parity_mode_t	1807
67.2.4	lpisci_bit_count_per_char_t	1807
67.2.5	lpisci_operation_config_t	1808
67.2.6	lpisci_receiver_source_t	1808
67.2.7	lpisci_wakeup_method_t	1808

Contents

Section Number	Title	Page Number
67.2.8	lpsci_idle_line_select_t	1808
67.2.9	lpsci_break_char_length_t	1808
67.2.10	lpsci_singlewire_txdir_t	1809
67.2.11	lpsci_ir_tx_pulsewidth_t	1809
67.2.12	lpsci_status_flag_t	1809
67.2.13	lpsci_interrupt_t	1810
67.3	Function Documentation	1810
67.3.1	LPSCI_HAL_Init	1810
67.3.2	LPSCI_HAL_EnableTransmitter	1810
67.3.3	LPSCI_HAL_DisableTransmitter	1811
67.3.4	LPSCI_HAL_IsTransmitterEnabled	1811
67.3.5	LPSCI_HAL_EnableReceiver	1811
67.3.6	LPSCI_HAL_DisableReceiver	1811
67.3.7	LPSCI_HAL_IsReceiverEnabled	1812
67.3.8	LPSCI_HAL_SetBaudRate	1812
67.3.9	LPSCI_HAL_SetBaudRateDivisor	1812
67.3.10	LPSCI_HAL_SetBitCountPerChar	1814
67.3.11	LPSCI_HAL_SetParityMode	1814
67.3.12	LPSCI_HAL_SetTxInversionCmd	1814
67.3.13	LPSCI_HAL_SetRxInversionCmd	1815
67.3.14	LPSCI_HAL_SetIntMode	1815
67.3.15	LPSCI_HAL_GetIntMode	1815
67.3.16	LPSCI_HAL_SetTxDataRegEmptyIntCmd	1815
67.3.17	LPSCI_HAL_GetTxDataRegEmptyIntCmd	1816
67.3.18	LPSCI_HAL_SetRxDataRegFullIntCmd	1816
67.3.19	LPSCI_HAL_GetRxDataRegFullIntCmd	1816
67.3.20	LPSCI_HAL_GetDataRegAddr	1817
67.3.21	LPSCI_HAL_Putchar	1817
67.3.22	LPSCI_HAL_Putchar9	1817
67.3.23	LPSCI_HAL_Putchar10	1817
67.3.24	LPSCI_HAL_Getchar	1817
67.3.25	LPSCI_HAL_Getchar9	1818
67.3.26	LPSCI_HAL_Getchar10	1818
67.3.27	LPSCI_HAL_SendDataPolling	1818
67.3.28	LPSCI_HAL_ReceiveDataPolling	1818
67.3.29	LPSCI_HAL_SetWaitModeOperation	1819
67.3.30	LPSCI_HAL_GetWaitModeOperation	1819
67.3.31	LPSCI_HAL_SetLoopCmd	1819
67.3.32	LPSCI_HAL_SetReceiverSource	1820
67.3.33	LPSCI_HAL_SetTransmitterDir	1820
67.3.34	LPSCI_HAL_PutReceiverInStandbyMode	1820
67.3.35	LPSCI_HAL_PutReceiverInNormalMode	1821
67.3.36	LPSCI_HAL_IsReceiverInStandby	1821
67.3.37	LPSCI_HAL_SetReceiverWakeupMethod	1821

Contents

Section Number	Title	Page Number
67.3.38	LPSCI_HAL_GetReceiverWakeupMethod	1822
67.3.39	LPSCI_HAL_ConfigIdleLineDetect	1822
67.3.40	LPSCI_HAL_SetBreakCharTransmitLength	1823
67.3.41	LPSCI_HAL_SetBreakCharDetectLength	1823
67.3.42	LPSCI_HAL_SetBreakCharCmd	1823
67.3.43	LPSCI_HAL_SetMatchAddress	1824
67.3.44	LPSCI_HAL_GetStatusFlag	1824
67.3.45	LPSCI_HAL_IsTxDataRegEmpty	1824
67.3.46	LPSCI_HAL_IsTxComplete	1825
67.3.47	LPSCI_HAL_IsRxDataRegFull	1825
67.3.48	LPSCI_HAL_ClearStatusFlag	1825
Chapter Vref_hal		
68.1	Overview	1827
68.2	Enumeration Type Documentation	1828
68.2.1	vref_mode_t	1828
68.2.2	vref_status_t	1828
68.3	Function Documentation	1828
68.3.1	VREF_HAL_Init	1828
68.3.2	VREF_HAL_SetVrefCmd	1828
68.3.3	VREF_HAL_IsVrefEnabled	1828
68.3.4	VREF_HAL_SetInternalRegulatorCmd	1829
68.3.5	VREF_HAL_IsInternalRegulatorEnabled	1829
68.3.6	VREF_HAL_SetTrimVal	1829
68.3.7	VREF_HAL_GetTrimVal	1830
68.3.8	VREF_HAL_IsInternalVoltRefStable	1830
68.3.9	VREF_HAL_SetBuffMode	1830
68.3.10	VREF_HAL_GetBuffMode	1830
Chapter Hwtimer_driver		
69.1	Overview	1833
69.2	Data Structure Documentation	1835
69.2.1	struct hwtimer_t	1835
69.2.2	struct hwtimer_time_t	1836
69.2.3	struct hwtimer_devif_t	1836
69.3	Enumeration Type Documentation	1837
69.3.1	_hwtimer_error_code_t	1837
69.4	Function Documentation	1837

Contents

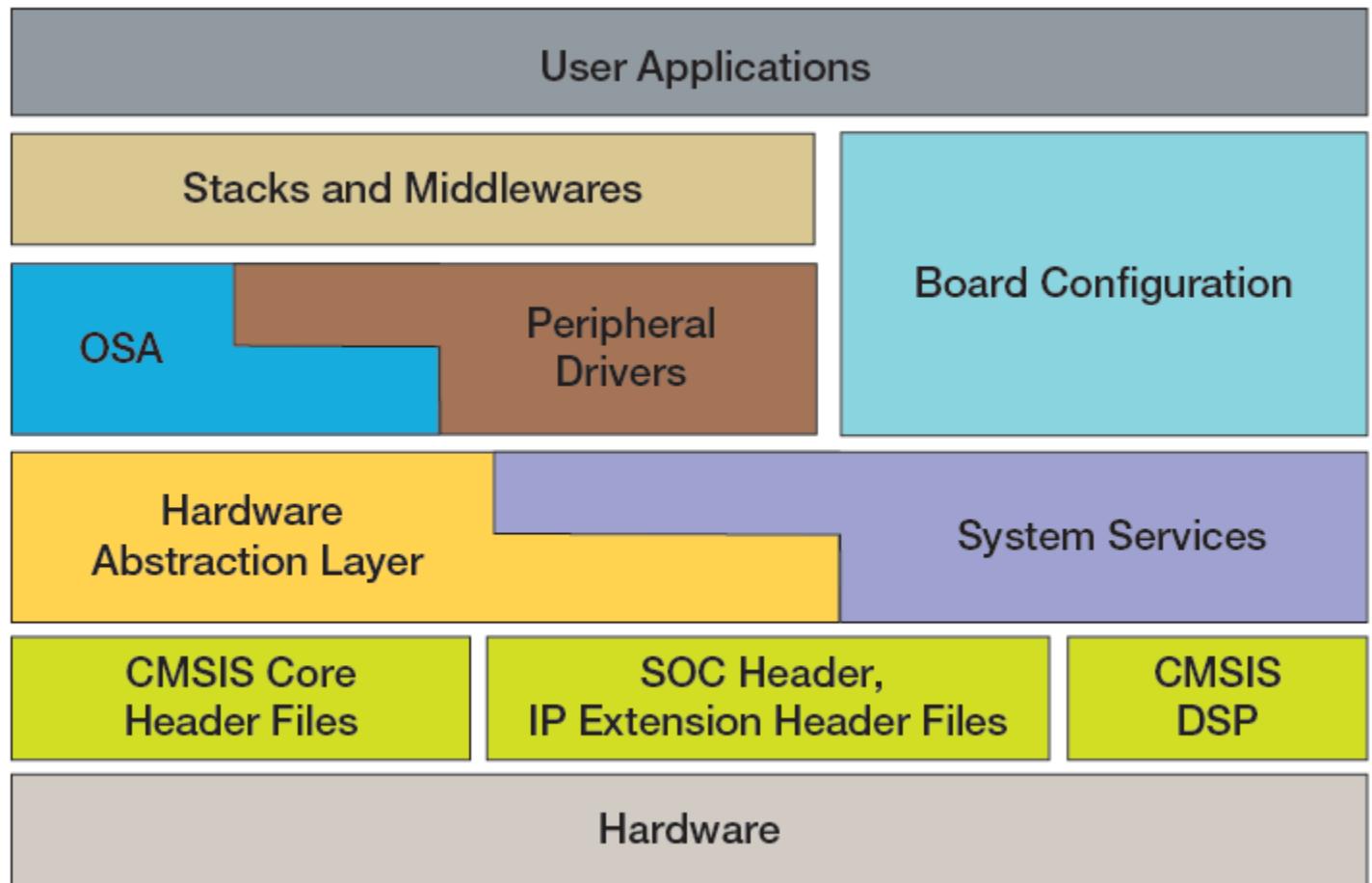
Section Number	Title	Page Number
69.4.1	HWTIMER_SYS_Init	1837
69.4.2	HWTIMER_SYS_Deinit	1838
69.4.3	HWTIMER_SYS_SetFreq	1839
69.4.4	HWTIMER_SYS_SetPeriod	1840
69.4.5	HWTIMER_SYS_GetFreq	1841
69.4.6	HWTIMER_SYS_GetPeriod	1842
69.4.7	HWTIMER_SYS_Start	1843
69.4.8	HWTIMER_SYS_Stop	1844
69.4.9	HWTIMER_SYS_GetModulo	1844
69.4.10	HWTIMER_SYS_GetTime	1845
69.4.11	HWTIMER_SYS_GetTicks	1846
69.4.12	HWTIMER_SYS_RegisterCallback	1847
69.4.13	HWTIMER_SYS_BlockCallback	1848
69.4.14	HWTIMER_SYS_UnblockCallback	1849
69.4.15	HWTIMER_SYS_CancelCallback	1850

Chapter 1

Introduction

The Kinetis software development kit (KSDK) is an extensive suite of robust peripheral drivers, stacks and middleware designed to simplify and accelerate application development on any Freescale Kinetis MCU. The addition of Processor Expert technology for software and board support configuration provides unmatched ease of use and flexibility. The Kinetis SDK is complimentary and includes full source code under a permissive open-source license for all hardware abstraction and peripheral driver software.

The Kinetis SDK consists of the following runtime software components written in C:



- ARM CMSIS Core and DSP standard libraries and CMSIS-compliant device header files which provide direct access to the peripheral registers and bits
- An open-source hardware abstraction layer (HAL) that provides a simple, stateless driver with an API encapsulating the low-level functions of the peripheral
- System services for centralized resources including a clock manager, interrupt manager, low-power manager, and a hardware timer

- Open-source, high-level peripheral drivers that build upon the HAL layer and may utilize one or more of the system services; drivers may be used as-is or as a reference for creating custom drivers
- An operating system abstraction (OSA) layer for adapting applications for use with a real time operating system (RTOS) or bare metal (no RTOS) applications. OSAs are provided for:
 - Freescale MQX™ RTOS
 - FreeRTOS
 - Micrium uC/OS-II
 - Micrium uC/OS-III
 - CMSIS-RTOS API compliant RTOS
 - bare-metal (no RTOS)
- Stacks and middleware in source or object formats including:
 - a comprehensive device and host USB stack with comprehensive USB class support
 - CMSIS DSP, a suite of common signal processing functions
 - FatFs, a FAT file system for small embedded systems
 - Encryption software utilizing the mmCAU hardware acceleration unit

The Kinetis SDK comes complete with software examples demonstrating the usage of the HAL, peripheral drivers, middleware, and RTOSes. All examples are provided with projects for the following toolchains:

- IAR Embedded Workbench
- Keil MDK
- Kinetis Design Studio
- GNU toolchain for ARM® Cortex®-M with makefile system

The HAL, peripheral drivers and system services can be used across multiple devices within the Kinetis product family without code modification. The configurable items for each driver, at all levels, are encapsulated into C language data structures. Kinetis devices specific configuration information is provided as part of the SDK and need not be modified by the user. HAL, peripheral driver, and system services configuration is not fixed and can be changed at runtime. Optionally, the entire driver set can be pre-built into a library.

The example applications demonstrate how to configure the drivers by passing configuration data to the APIs. In addition to the software source, Processor Expert is provided as a time-saving option for software configuration. Processor Expert is a complimentary PC-hosted software configuration tool (Eclipse plugin) with complete knowledge of all Kinetis MCUs. It provides a graphical user interface to handle MCU-specific board configuration and driver tuning tasks including:

- Optional generation of low-level device initialization code for post-reset configuration
- Package I/O allocation and pin initialization source code generation
- Creation and management of HAL and peripheral driver C source configuration data structures

The organization of files in the Kinetis SDK release package is focused on ease-of-use. The Kinetis SDK folder hierarchy is organized at the top level with these folders:

platform	Platform folder contains code for SDK HAL and Peripheral drivers. The folder contain following sub-folders								
drivers	<p>Peripheral drivers for every peripheral supported For each peripheral there is a sub-folder under it containing following files/folders:</p> <table border="1"> <tr> <td>Source</td><td>Source folder containing .c files for the driver code</td></tr> <tr> <td>fsl_ipname_driver.h</td><td>Peripheral driver API header file</td></tr> </table>			Source	Source folder containing .c files for the driver code	fsl_ipname_driver.h	Peripheral driver API header file		
Source	Source folder containing .c files for the driver code								
fsl_ipname_driver.h	Peripheral driver API header file								
hal	<p>HAL drivers for every peripheral supported For each peripheral there is a sub-folder under it containing following files:</p> <table border="1"> <tr> <td>fsl_ipname_hal.c</td><td>HAL driver source code file</td></tr> <tr> <td>fsl_ipname_hal.h</td><td>HAL driver API header file</td></tr> <tr> <td>fsl_ipname_features.h</td><td>Features header file for the IP defining features supported for specific chipset</td></tr> </table>			fsl_ipname_hal.c	HAL driver source code file	fsl_ipname_hal.h	HAL driver API header file	fsl_ipname_features.h	Features header file for the IP defining features supported for specific chipset
fsl_ipname_hal.c	HAL driver source code file								
fsl_ipname_hal.h	HAL driver API header file								
fsl_ipname_features.h	Features header file for the IP defining features supported for specific chipset								
include	Platform specific headers and board specific CMSIS header files								
startup	Chip specific CMSIS compliant startup code								
utilities	Tools								
linker	Board specific linker files								
apps	Contain demo applications code								
doc	User Guide and Quick start guides for each chip supported								
boards	Board specific code								
lib	Prebuilt libraries for each tool chain and boards supported								
mk	Common make files used for compiling with GCC								
rtos	RTOS abstraction layer code for supported RTOSes								
usb	Freescale's USB stack code								

The rest of the document describes the API references in detail for HAL and Peripheral drivers.

Chapter 2

Architectural Overview

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

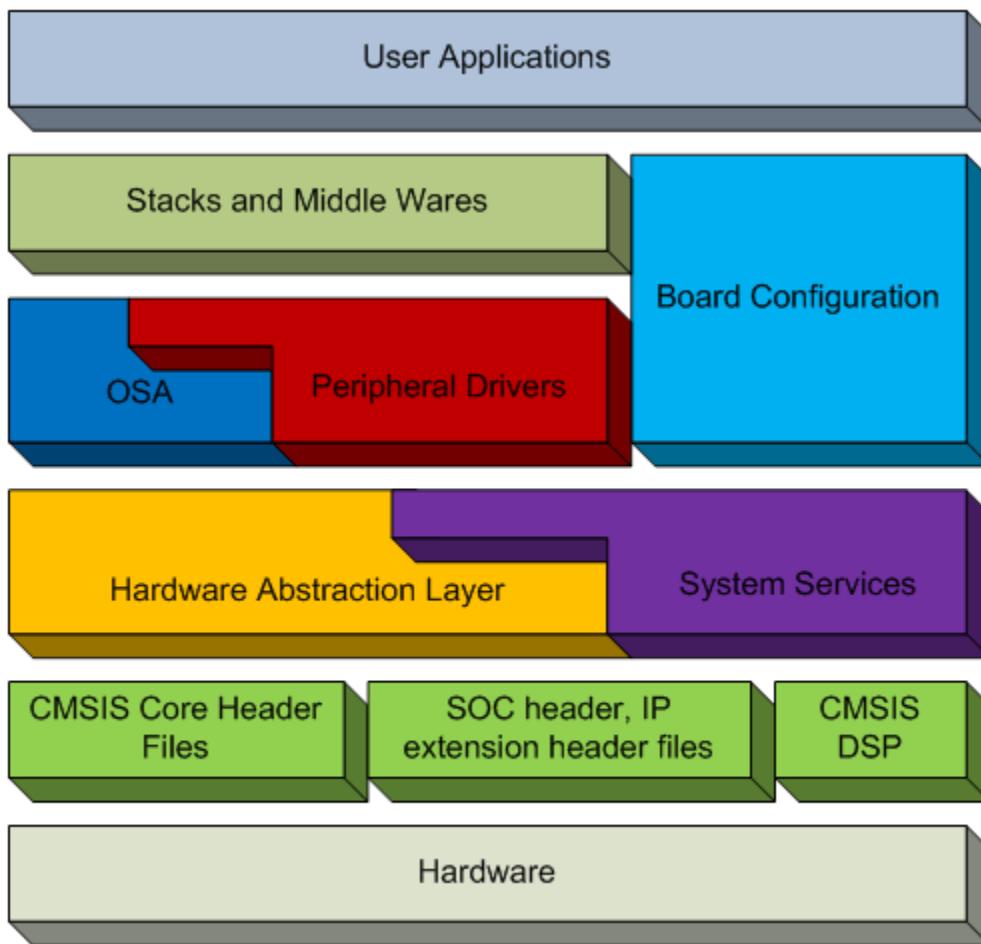
Overview

The KSDK architecture consists of eight key components listed below.

1. The ARM Cortex Microcontroller Software Interface Standard (CMSIS) CORE compliance device specific header files
2. System Services
3. Hardware Abstraction Layer
4. Peripheral Driver Layer
5. Real-time Operating System (RTOS) Abstraction Layer
6. Board-specific configuration
7. Stack and Middleware that integrate with KSDK
8. Applications based on the KSDK architecture

This image shows how each component stacks up.

The Kinetis SDK consists of these runtime software components written in C:



Kinetis MCU header files

The KSDK contains CMSIS-compliant device-specific header files which provide direct access to the Kinetis MCU peripheral registers. Each supported Kinetis MCU device in KSDK 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 an access to the peripheral registers through pointers and predefined masks.

In addition to the overall SoC memory-mapped header file, the KSDK includes extension header files for each peripheral instantiated on the Kinetis MCU. These peripheral extension header files define C union data structures that represent each register bit field within a peripheral for convenient register accesses. The extension header files also take advantage of the bit band feature when reading from or writing to 1-bit wide fields within a register. It also provides register address offsets for each instance of the peripheral along with address offset for each register within an instance. When accessing the register macros in the extension header file, the user must pass in the register base address for the desired peripheral. The KSDK HAL Driver (discussed later) API functions take in the base address and then passes this into the extension header file macros for peripheral register accesses.

Along with the SoC header files and peripheral extension header files, the KSDK also includes common CMSIS header files for the ARM Cortex-M core and DSP library from the latest CMSIS release. The CMSIS DSP library source code is also included for reference. These files and the above mentioned header

files can all be found in the KSDK platform/CMSIS directory.

System Services

The KSDK System Services contains a set of software entities that can be used by the Peripheral Drivers. They may be used with HAL Drivers to build the Peripheral Drivers or they can be used by an application directly. The following sections describe each of the System Services software entities. These System Services are in the KSDK platform/system directory.

Interrupt Manager

The Interrupt Manager provides functions to enable and disable individual interrupts within the Nested Vector Interrupt Controller (NVIC). It also provides functions to enable and disable the ARM core global interrupt (via the CPSIE and CPSID instructions) for bare-metal critical section implementation. In addition to providing functions for interrupt enabling and disabling, the Interrupt Manager provides Interrupt Service Routine (ISR) registration that allows the application software to register or replace the interrupt handler for a specified IRQ vector. The KSDK drivers do not set interrupt priorities. The interrupt priority scheme is entirely determined by the specific application logic and its setting is handled by the user application. The user application manages the interrupt priorities by using the NVIC functions provided in the ARM Cortex-M core CMSIS header file.

Clock Manager

The Clock Manager provides centralized clock-related functions for the entire system. It can dynamically set the system clock and perform clock gating/un-gating for specific peripherals. The Clock Manager also maintains knowledge of the clock sources required for each peripheral and provides functions to obtain the clock frequency for each supported clock used by the peripheral.

Unified Hardware (HW) Timer

The Unified HW Timer provides a common timer interface that can be linked with any supported Kinetis MCU hardware timer peripheral or with the ARM core system timer (SysTick) to perform basic timer operations. It can be used as a shared timer capable of microsecond resolution for bare-metal use-cases, such as to time guard busy loops. It can also be used when interfacing with an RTOS to provide operating system (OS) time ticks. Future implementations of the Unified HW Timer will take advantage of low power timer peripherals that will allow OS ticks to continue when the system is in various lower power modes.

Hardware Abstraction Layer (HAL)

The KSDK HAL consists of low-level drivers for the Kinetis MCU product family on-chip peripherals. The main goal is to abstract the hardware peripheral register accesses into a set of stateless basic functional operations. The HAL itself can be used with system services to build application-specific logic or as building blocks for use-case driven high-level Peripheral Drivers. It primarily focuses on the functional control, configuration, and realization of basic peripheral operations. The HAL hides register access details and various MCU peripheral instantiation differences so that, either an application or high-level Peripheral Drivers, can be abstracted from the low-level HW details. Therefore, hardware peripheral must be accessed through HAL.

The HAL can also support some high-level functions with certain logic, provided these high-level functions do not depend on functions from other peripherals, nor impose any action to be taken in interrupt

service routines. For example, the UART HAL provides a blocking byte-send function that relies only on the features available in the UART peripheral itself. These high-level functions enhance the usability of the HAL but remain stateless and independent of other peripherals. Essentially, the HAL functional boundary is limited by the peripheral itself. There is one HAL driver for each peripheral and the HAL only accesses the features available within the peripheral. In addition, the HAL does not define interrupt service routine entries or support interrupt handling. These tasks must be handled by a high-level Peripheral Driver together with the Interrupt Manager.

The HAL drivers can be found in the KSDK platform/hal directory.

Feature Header Files

The HAL is designed to be reusable regardless of the peripheral configuration differences from one Kinetis MCU device to another. A Peripheral Feature Header File is provided for each peripheral type to define the feature or configuration differences for each Kinetis sub-family device. The feature header files are integrated with the HAL so that it is tailored to the exact feature or configuration supported for a particular device.

Design Guidelines

This section summarizes the design guidelines to develop the HAL drivers. It is meant for information purposes and provides more details on the make-up of the HAL drivers. As previously stated, the main goal of the HAL is to abstract the hardware details and provide a set of easy-to-use low-level drivers. The HAL itself can be used directly by the user application; in this case, the high-level Peripheral Drivers that are built on top of the HAL serve as a reference implementation and to demonstrate the HAL usage. The HAL is mainly focused on individual functional primitives and makes no assumption of use-cases. It also implements some high-level functions with certain logic without dependencies from other peripherals and does not impose any interrupt handling. The HAL APIs follow a naming convention that is consistent from one peripheral to the next. This is a summary of the design guidelines used when developing the HAL drivers:

- There is a dedicated HAL driver for each individual peripheral. Peripherals with different versions or configurations are treated as the same peripheral with differences handled through a feature header file. HAL should hide both the peripheral and MCU details and differences from the high-level application and drivers.
- Each HAL driver has an initialization function to put the peripheral into a known default state (i.e. the peripheral reset state). There is no corresponding de-initialization function.
- Each HAL driver has an enable and disable function to enable or disable the peripheral module.
- The HAL driver does not have an internal operation context and should not dynamically allocate memory.
- The HAL driver does not take in any configuration data from a high-level driver or high-level application because the HAL does not assume any use-cases.
- The HAL provides both blocking and non-blocking functions for peripherals that support data transaction-related operations.
- The HAL may implement high-level functions based on abstracted functional primitives, provided this implementation does not depend on functions outside of the peripheral and does not involve interrupt handling. The HAL must remain stateless.
- The HAL driver does not depend on any other software entities or invoke any functions from other

peripherals.

Peripheral Drivers

The KSDK Peripheral Drivers are use-case driven high-level drivers that implement high-level logic transactions based on one or more HAL drivers, other Peripheral Drivers, and/or System Services. Consider, for example, the differences in the UART HAL and the UART Peripheral Driver. The UART HAL mainly focuses on byte-level basic functional primitives, while the UART Peripheral Driver operates on an interrupt-driven level using data buffers to transfer a stream of bytes and may be able to interface with DMA Peripheral Driver for DMA-enabled transfers between data buffers. In general, if a driver, that is mainly based on one peripheral, interfaces with functions beyond its own HAL and/or requires interrupt servicing, the driver is considered a high-level Peripheral Driver.

The KSDK Peripheral Drivers support all instances of each peripheral instantiated on the Kinetis MCU by using a simple integer parameter for the peripheral instance number. Each Peripheral Driver includes a “common” file, denoted as `fsl_<peripheral>_common.c` (where `<peripheral>` is the name of the peripheral for which the driver is written). This file contains the translation of the peripheral instance number to the peripheral register base address, which is then passed into the HAL. Hence, the user of the Peripheral Driver does not need to know the peripheral memory-mapped base address.

The Peripheral Drivers operate on a 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 for the driver internal operation through the driver initialization function.

The Peripheral Drivers are designed to handle the entire functionality for a targeted use-case. An application should be able to use only the Peripheral Driver to accomplish its purpose. The mixing of the Peripheral Driver and HAL by an application for the same peripheral can be done, but is discouraged for architectural cleanliness and to avoid cases where bypassing the Peripheral Driver results in logic errors within the Peripheral Driver.

The Peripheral Drivers can be found in the KSDK platform/drivers directory.

Interrupt Handling

Interrupt-driven Peripheral Drivers are designed to service all interrupts for their desired functionality. Each Peripheral Driver implements a set of easy-to-use APIs for a particular use-case. Each Peripheral Driver also exposes interrupt functions that implement all actions taken when serving a particular interrupt. These interrupt action functions take in the peripheral instance number as an input parameter. All Interrupt Service Routine (ISR) entries for the Peripheral Driver are implemented in a separate C file named the `fsl_<peripheral>_irq.c`, which is located in the top level of the Peripheral Driver folder. The ISR entries invoke a corresponding interrupt action functions that are implemented in the driver and are exposed as part of the driver public interfaces. Each interrupt action function is hard coded with the peripheral instance number. Note that the IRQ file depends on the Peripheral Driver but the driver does not depend on the IRQ file.

The CMSIS startup code contains the vector table with the ISR entry names which are defined as weak references. The Peripheral Driver ISR entry names match the names defined in the vector table such that these newly-defined ISR entries in the driver replace the weak references defined in the vector table. There is no dependency on the location of the vector table. However, if the vector table is re-located (normally to RAM), the ISR entry names should remain consistent.

Each Peripheral Driver IRQ file defines the ISR entries for all instances of the peripheral instantiated on the MCU. Any unused ISR entries can be safely deleted in the user application. Note that, when building driver libraries for a particular MCU, the IRQ files are not part of the library build. When developing an application, however, include these IRQ files into the application project space.

If you want to use the HAL to directly build an interrupt-driven application or a high-level driver, define the ISR entries to service needed interrupts. The ISR entry names have to match the names of the ISR entry names provided in the CMSIS startup code vector table. Normally, the vector table is relocated to RAM during system startup and the ISR entry names remain unchanged. However, should these ISR entry names change, the user is required to register the newly defined ISR entry table names by invoking the Interrupt Manager registration function.

When working with an RTOS, if the target operating system has its own interrupt handling mechanism, the user's ISR has to dynamically register the internal vector table maintained inside the OS via the service routine provided by the OS. Some operating systems do not have dedicated interrupt handling, and no special handling is required. Most of the RTOSes designed for the Cortex-M core use PendSV exceptions for scheduling. The PendSV exceptions are low priority and execute only when all other interrupts have been serviced. There are special cases where interrupt handling prologues and epilogues have to be called at the beginning and at the end of an ISR to inform the OS of the entry and exit of an ISR so that the OS scheduler does not trigger a rescheduling event while the system is servicing interrupts. The OS Abstraction layer provides prologues and epilogues in a uniform API for all supported OS.

Peripheral Driver Types

Peripheral Drivers are designed and operated on use-case driven high-level logic. For that reason, the interaction between a Peripheral driver and other software components has a few limitations. This is a list of typical KSDK Peripheral Driver types to demonstrate this concept:

- Peripheral Drivers that use one HAL for a particular IP and system services
- Peripheral Drivers that use multiple HAL components and System Services. An example involves a Peripheral Driver that uses both the DMA HAL and the I2C HAL to build a DMA-enabled high-level I2C driver. At the same time, a DMA Peripheral Driver already exists and provides the overall DMA channel management. In this situation, issues can arise for the dedicated DMA Peripheral Driver because it is unaware of use of resources for the DMA-enabled I2C driver that accesses the DMA HAL directly. In this case, the KSDK Peripheral Drivers will only use the DMA HAL when configuring the DMA Transfer Control Descriptor for a particular channel that is dedicated to the driver and avoid using the DMA HAL to change the DMA peripheral control registers. The DMA channel allocation for a particular Peripheral Driver is normally assigned during the initialization of the peripheral by calling the DMA Peripheral Driver channel request function. In this way, the DMA maintains a used channel registry.
- Peripheral Drivers that use a combination of HAL, other Peripheral Drivers, and System Services for solution-based composite high level drivers.

Design guidelines

This section summarizes the design guidelines to develop the Peripheral Drivers. In this list, the term Peripheral Driver is replaced with the acronym “PD”:

- The PD does not directly access hardware registers and only uses HAL to interface with hardware.

It uses other PDs indirectly to access hardware functions.

- The PD does not dynamically allocate memory and does not assume any static configuration at compile-time. It supports run-time configurations by taking in a configuration data structure from the Application in an initialization function. This initialization function also takes in the memory allocated for internal operational context storage needs. The PD only reads from the configuration data structure. The configuration data structure is defined as “const”.
- The PD supports an initialization function and has a corresponding de-initialization function.
- The PD supports both blocking and non-blocking functions for data transactions.
- The PD does not depend on any software or hardware entities that do not relate to any of the peripherals available on the Kinetis platform. For example, the Ethernet (ENET) PD does not depend on an external ENET PHY, even if the ENET PD does not function without an associated external PHY.
- If the PD is built on top of HAL, the PD can work with any instance of the peripheral by taking the instance number as a parameter. When global data is shared across multiple instances, the global data access has to be protected when working with an RTOS. This can be done using the critical section functions implemented as part of the RTOS abstraction. This data access protection is addressed with the bare-metal OS abstraction implementation when no RTOS is used.
- When the ISR and other public API functions within the driver access the PD internal operation context, that data must be defined as volatile so that compiler does not inadvertently optimize the code resulting in an undesired functionality.
- The PD provides enough functions to allow the APIs provided by PD to meet the target use-case implementation. The user application should not mix the use of both a PD and HAL .
- The PD does not configure pin muxing or set up interrupt priorities for related interrupts. Those items are handled by the board-specific configuration and the application-specific logic.

Demo Applications

The Demo Applications provided in the KSDK provide examples, which show how to build user applications using the KSDK framework. The Demo Applications can be found in the KSDK top-level demo directory. The KSDK includes two types of demo applications:

- Demo Applications that demonstrate the usage of the Peripheral Drivers.
- Demo Applications that provide a reference design based on the features available on the target Kinetis MCU and evaluation boards. This type of demo is targeted to highlight a certain feature of the SOC for its intended usage, and/or to provide turnkey references using the KSDK driver library with other integrated software components such as USB stacks.

Board Configuration

The KSDK drivers make no assumption on board-specific configurations nor do the drivers configure pin muxing, which are part of the board-specific configuration. The KSDK provides board-configuration files that un-gate clocks for related I/O ports, configure pin muxing for the entire board, and functions that can be called before driver initialization. These board-configuration files can be found in the KSDK top-level board directory.

In addition, the KSDK also contains a set of drivers in the boards/common directory for common devices (such as SPI flash and ENET PHY) found on the Kinetis platform evaluation boards. These drivers are included mainly for a convenient out-of-box experience and to demonstrate how to build use-case-driven

reference designs.

OS Abstraction layer

The KSDK drivers are designed to work with or without an RTOS. The Operating System Abstraction layer (OSA) is designed and implemented for supported RTOS to provide a common set of service routines for drivers, integrated software solution, and upper-level applications so that a common code base can be used regardless the target OS.

The services supported by the OSA are driven by the driver requirements, supported middleware, and applications. The OSA layer is designed to be as thin as possible. The OSA either maps the desired services to the services provided by the target OS or implements the services that the target OS does not support.

The OSA itself does not dynamically allocate memory for implementing an OS service component. Instead, the memory for OS components is allocated by the caller and is passed into the OSA for servicing. The OSA drivers can be found in the KSDK platform/osa directory.

Software Stacks and other Middleware integration

The core of the Kinetis SDK is a set of common driver libraries built for all Kinetis MCU product family peripherals. Kinetis SDK also provides a foundation for software stacks and other middleware. The KSD-K integrates other Freescale or third party enablement software stacks, such as a USB stack and a TCP/IP stack and other middleware, to offer a complete, easy-to-use, software development kit to the Kinetis MCU users. The KSDK framework integrates all Kinetis MCU software solutions for the Kinetis product families.

Chapter 3

16-bit SAR Analog-to-Digital Converter (ADC16)

3.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the 16-bit SAR Analog-to-Digital Converter (ADC16) block of Kinetis devices.

Modules

- [ADC16 HAL Driver](#)
- [ADC16 Peripheral Driver](#)

3.2 ADC16 HAL Driver

3.2.1 Overview

This section describes the programming interface of the ADC16 HAL driver.

Enumerations

- enum `adc16_status_t` {
 `kStatus_ADC16_Success` = 0U,
 `kStatus_ADC16_InvalidArgument` = 1U,
 `kStatus_ADC16_Failed` = 2U }
 ADC16 status return codes.
- enum `adc16_clk_divider_mode_t` {
 `kAdcClkDividerInputOf1` = 0U,
 `kAdcClkDividerInputOf2` = 1U,
 `kAdcClkDividerInputOf4` = 2U,
 `kAdcClkDividerInputOf8` = 3U }
 Defines the type of the enumerating divider for the converter.
- enum `adc16_resolution_mode_t` {
 `kAdcResolutionBitOf8or9` = 0U,
 `kAdcResolutionBitOfSingleEndAs8` = `kAdcResolutionBitOf8or9`,
 `kAdcResolutionBitOfDiffModeAs9` = `kAdcResolutionBitOf8or9`,
 `kAdcResolutionBitOf12or13` = 1U,
 `kAdcResolutionBitOfSingleEndAs12` = `kAdcResolutionBitOf12or13`,
 `kAdcResolutionBitOfDiffModeAs13` = `kAdcResolutionBitOf12or13`,
 `kAdcResolutionBitOf10or11` = 2U,
 `kAdcResolutionBitOfSingleEndAs10` = `kAdcResolutionBitOf10or11`,
 `kAdcResolutionBitOfDiffModeAs11` = `kAdcResolutionBitOf10or11` }
 Defines the type of the enumerating resolution for the converter.
- enum `adc16_clk_src_mode_t` {
 `kAdcClkSrcOfBusClk` = 0U,
 `kAdcClkSrcOfBusOrAltClk2` = 1U,
 `kAdcClkSrcOfAltClk` = 2U,
 `kAdcClkSrcOfAsynClk` = 3U }
 Defines the type of the enumerating source of the input clock.
- enum `adc16_long_sample_cycle_mode_t` {
 `kAdcLongSampleCycleOf24` = 0U,
 `kAdcLongSampleCycleOf16` = 1U,
 `kAdcLongSampleCycleOf10` = 2U,
 `kAdcLongSampleCycleOf4` = 3U }
 Defines the type of the enumerating long sample cycles.
- enum `adc16_ref_volt_src_mode_t` {
 `kAdcRefVoltSrcOfVref` = 0U,
 `kAdcRefVoltSrcOfValt` = 1U }
 Defines the type of the enumerating reference voltage source.

- enum `adc16_hw_cmp_range_mode_t` {

 `kAdcHwCmpRangeModeOf1` = 0U,

 `kAdcHwCmpRangeModeOf2` = 1U,

 `kAdcHwCmpRangeModeOf3` = 2U,

 `kAdcHwCmpRangeModeOf4` = 3U }

Defines the type of the enumerating asserted range in the hardware compare.

Functions

- void `ADC16_HAL_Init` (`uint32_t` baseAddr)

Resets all registers into a known state for the ADC16 module.
- static void `ADC16_HAL_ConfigChn` (`uint32_t` baseAddr, `uint32_t` chnGroup, `bool` intEnable, `bool` diffEnable, `uint8_t` chnNum)

Configures the conversion channel for the ADC16 module.
- static `bool` `ADC16_HAL_GetChnConvCompletedCmd` (`uint32_t` baseAddr, `uint32_t` chnGroup)

Checks whether the channel conversion is completed.
- static void `ADC16_HAL_SetLowPowerCmd` (`uint32_t` baseAddr, `bool` enable)

Switches to enable the low power mode for ADC16 module.
- static void `ADC16_HAL_SetClkDividerMode` (`uint32_t` baseAddr, `adc16_clk_divider_mode_t` mode)

Selects the clock divider mode for the ADC16 module.
- static void `ADC16_HAL_SetLongSampleCmd` (`uint32_t` baseAddr, `bool` enable)

Switches to enable the long sample mode for the ADC16 module.
- static void `ADC16_HAL_SetResolutionMode` (`uint32_t` baseAddr, `adc16_resolution_mode_t` mode)

Selects the conversion resolution mode for ADC16 module.
- static `adc16_resolution_mode_t` `ADC16_HAL_GetResolutionMode` (`uint32_t` baseAddr)

Gets the conversion resolution mode for ADC16 module.
- static void `ADC16_HAL_SetClkSrcMode` (`uint32_t` baseAddr, `adc16_clk_src_mode_t` mode)

Selects the input clock source for the ADC16 module.
- static void `ADC16_HAL_SetAsyncClkCmd` (`uint32_t` baseAddr, `bool` enable)

Switches to enable the asynchronous clock for the ADC16 module.
- static void `ADC16_HAL_SetHighSpeedCmd` (`uint32_t` baseAddr, `bool` enable)

Switches to enable the high speed mode for the ADC16 module.
- static void `ADC16_HAL_SetLongSampleCycleMode` (`uint32_t` baseAddr, `adc16_long_sample_cycle_mode_t` mode)

Selects the long sample cycle mode for the ADC16 module.
- static `uint16_t` `ADC16_HAL_GetChnConvValueRAW` (`uint32_t` baseAddr, `uint32_t` chnGroup)

Gets the raw result data of channel conversion for the ADC16 module.
- static void `ADC16_HAL_SetHwCmpValue1` (`uint32_t` baseAddr, `uint16_t` value)

Sets the compare value of the lower limitation for the ADC16 module.
- static void `ADC16_HAL_SetHwCmpValue2` (`uint32_t` baseAddr, `uint16_t` value)

Sets the compare value of the higher limitation for the ADC16 module.
- static `bool` `ADC16_HAL_GetConvActiveCmd` (`uint32_t` baseAddr)

Checks whether the converter is active for the ADC16 module.
- static void `ADC16_HAL_SetHwTriggerCmd` (`uint32_t` baseAddr, `bool` enable)

Switches to enable the hardware trigger mode for the ADC16 module.
- static void `ADC16_HAL_SetHwCmpCmd` (`uint32_t` baseAddr, `bool` enable)

Switches to enable the hardware comparator for the ADC16 module.

ADC16 HAL Driver

- static void [ADC16_HAL_SetHwCmpGreaterCmd](#) (uint32_t baseAddr, bool enable)
Switches to enable the setting that is greater than the hardware comparator.
- static void [ADC16_HAL_SetHwCmpRangeCmd](#) (uint32_t baseAddr, bool enable)
Switches to enable the setting of the range for hardware comparator.
- void [ADC16_HAL_SetHwCmpMode](#) (uint32_t baseAddr, [adc16_hw_cmp_range_mode_t](#) mode)
Configures the asserted range of the hardware comparator for the ADC16 module.
- static void [ADC16_HAL_SetRefVoltSrcMode](#) (uint32_t baseAddr, [adc16_ref_volt_src_mode_t](#) mode)
Selects the reference voltage source for the ADC16 module.
- static void [ADC16_HAL_SetContinuousConvCmd](#) (uint32_t baseAddr, bool enable)
Switches to enable the continuous conversion mode for the ADC16 module.

3.2.2 Enumeration Type Documentation

3.2.2.1 enum adc16_status_t

Enumerator

- kStatus_ADC16_Success* Success.
kStatus_ADC16_InvalidArgument Invalid argument existed.
kStatus_ADC16_Failed Execution failed.

3.2.2.2 enum adc16_clk_divider_mode_t

Enumerator

- kAdcClkDividerInputOf1* For divider 1 from the input clock to ADC.
kAdcClkDividerInputOf2 For divider 2 from the input clock to ADC.
kAdcClkDividerInputOf4 For divider 4 from the input clock to ADC.
kAdcClkDividerInputOf8 For divider 8 from the input clock to ADC.

3.2.2.3 enum adc16_resolution_mode_t

Enumerator

- kAdcResolutionBitOf8or9* 8-bit for single end sample, or 9-bit for differential sample.
kAdcResolutionBitOfSingleEndAs8 8-bit for single end sample.
kAdcResolutionBitOfDiffModeAs9 9-bit for differential sample.
kAdcResolutionBitOf12or13 12-bit for single end sample, or 13-bit for differential sample.
kAdcResolutionBitOfSingleEndAs12 12-bit for single end sample.
kAdcResolutionBitOfDiffModeAs13 13-bit for differential sample.
kAdcResolutionBitOf10or11 10-bit for single end sample, or 11-bit for differential sample.
kAdcResolutionBitOfSingleEndAs10 10-bit for single end sample.
kAdcResolutionBitOfDiffModeAs11 11-bit for differential sample.

3.2.2.4 enum adc16_clk_src_mode_t

Enumerator

- kAdcClkSrcOfBusClk* For input as bus clock.
- kAdcClkSrcOfBusOrAltClk2* For input as bus clock /2 or AltClk2.
- kAdcClkSrcOfAltClk* For input as alternate clock (ALTCLK).
- kAdcClkSrcOfAsynClk* For input as asynchronous clock (ADACK).

3.2.2.5 enum adc16_long_sample_cycle_mode_t

Enumerator

- kAdcLongSampleCycleOf24* 20 extra ADCK cycles, 24 ADCK cycles total.
- kAdcLongSampleCycleOf16* 12 extra ADCK cycles, 16 ADCK cycles total.
- kAdcLongSampleCycleOf10* 6 extra ADCK cycles, 10 ADCK cycles total.
- kAdcLongSampleCycleOf4* 2 extra ADCK cycles, 6 ADCK cycles total.

3.2.2.6 enum adc16_ref_volt_src_mode_t

Enumerator

- kAdcRefVoltSrcOfVref* For external pins pair of VrefH and VrefL.
- kAdcRefVoltSrcOfValt* For alternate reference pair of ValtH and ValtL.

3.2.2.7 enum adc16_hw_cmp_range_mode_t

When the internal CMP is enabled, the COCO flag, which represents the complement of the conversion, is not asserted if the sample value is not in the indicated range. Eventually, the data of conversion result is not kept in the result data register. The two values, cmpValue1 and cmpValue2, mark the thresholds with the comparator feature.

- kAdcHwCmpRangeModeOf1: Both greater than and in range switchers are disabled. The available range is "< cmpValue1".
- kAdcHwCmpRangeModeOf2: Greater than switcher is enabled while the in range switcher is disabled. The available range is "> cmpValue1".
- kAdcHwCmpRangeModeOf3: Greater than switcher is disabled while in range switcher is enabled. The available range is "< cmpValue1" or "> cmpValue2" when cmpValue1 <= cmpValue2, or "< cmpValue1" and "> cmpValue2" when cmpValue1 >= cmpValue2.
- kAdcHwCmpRangeModeOf4: Both greater than and in range switchers are enabled. The available range is "> cmpValue1" and "< cmpValue2" when cmpValue1 <= cmpValue2, or "> cmpValue1" or "< cmpValue2" when cmpValue1 < cmpValue2.

ADC16 HAL Driver

Enumerator

- kAdcHwCmpRangeModeOf1*** For selection mode 1.
- kAdcHwCmpRangeModeOf2*** For selection mode 2.
- kAdcHwCmpRangeModeOf3*** For selection mode 3.
- kAdcHwCmpRangeModeOf4*** For selection mode 4.

3.2.3 Function Documentation

3.2.3.1 void ADC16_HAL_Init (*uint32_t baseAddr*)

This function resets all registers into a known state for the ADC module. This known state is the reset value indicated by the Reference manual. It is strongly recommended to call this API before any other operation when initializing the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

3.2.3.2 static void ADC16_HAL_ConfigChn (*uint32_t baseAddr, uint32_t chnGroup, bool intEnable, bool diffEnable, uint8_t chnNum*) [inline], [static]

This function configures the channel for the ADC16 module. At any point, only one of the configuration groups takes effect. The other channel mux of the first group (group A, 0) is only for the hardware trigger. Both software and hardware trigger can be used to the first group. When in software trigger mode, once the available channel is set, the conversion begins to execute.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chnGroup</i>	Channel configuration group ID.
<i>intEnable</i>	Switcher to enable interrupt when conversion is completed.
<i>diffEnable</i>	Switcher to enable differential channel mode.
<i>chnNum</i>	ADC16 channel for next conversion.

3.2.3.3 static bool ADC16_HAL_GetChnConvCompletedCmd (*uint32_t baseAddr, uint32_t chnGroup*) [inline], [static]

This function checks whether the channel conversion for the ADC module is completed.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chnGroup</i>	Channel configuration group ID.

Returns

Assertion of completed conversion mode.

3.2.3.4 static void ADC16_HAL_SetLowPowerCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the low power mode for ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.5 static void ADC16_HAL_SetClkDividerMode (*uint32_t baseAddr, adc16_clk_divider_mode_t mode*) [inline], [static]

This function selects the clock divider mode for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode enumeration. See to "adc16_clk_divider_mode_t".

3.2.3.6 static void ADC16_HAL_SetLongSampleCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the long sample mode for the ADC16 module. This function adjusts the sample period to allow the higher impedance inputs to be accurately sampled or to maximize the conversion speed for the lower impedance inputs. Longer sample times can also be used to lower overall power consumption if the continuous conversions are enabled and the high conversion rates are not required. If the long sample mode is enabled, more configuration is set by calling the "ADC16_HAL_SetLongSampleCycleMode()" function.

ADC16 HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.7 static void ADC16_HAL_SetResolutionMode (uint32_t *baseAddr*, adc16_resolution_mode_t *mode*) [inline], [static]

This function selects the conversion resolution mode for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode enumeration. See to "adc16_resolution_mode_t".

3.2.3.8 static adc16_resolution_mode_t ADC16_HAL_GetResolutionMode (uint32_t *baseAddr*) [inline], [static]

This function gets the conversion resolution mode for the ADC16 module. It is specially used when processing the conversion result of RAW format.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

Current conversion resolution mode.

3.2.3.9 static void ADC16_HAL_SetClkSrcMode (uint32_t *baseAddr*, adc16_clk_src_mode_t *mode*) [inline], [static]

This function selects the input clock source for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode enumeration. See to "adc16_clk_src_mode_t".

3.2.3.10 static void ADC16_HAL_SetAsyncClkCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the asynchronous clock for the ADC16 module. It enables the ADC's asynchronous clock source and the clock source output regardless of the conversion and the input clock select status of the ADC. Asserting this function allows the clock to be used even while the ADC is idle or operating from a different clock source. Also, latency of initiating a single or first-continuous conversion with the asynchronous clock selected is reduced since the ADC16 internal clock has been already operational.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.11 static void ADC16_HAL_SetHighSpeedCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the high speed mode for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.12 static void ADC16_HAL_SetLongSampleCycleMode (uint32_t *baseAddr*, adc16_long_sample_cycle_mode_t *mode*) [inline], [static]

This function selects the long sample cycle mode for the ADC16 module. This function should be called along with "ADC16_HAL_SetLongSampleCmd()".

Parameters

ADC16 HAL Driver

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of long sample cycle mode. See the "adc16_long_sample_cycle_mode_t".

3.2.3.13 static uint16_t ADC16_HAL_GetChnConvValueRAW (uint32_t *baseAddr*, uint32_t *chnGroup*) [inline], [static]

This function gets the result data of conversion for the ADC16 module. The return value is raw data that is not processed. The unavailable bits would be filled with "0" in single-ended mode and sign bit in differential mode.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chnGroup</i>	Channel configuration group ID.

Returns

Conversion value of RAW.

3.2.3.14 static void ADC16_HAL_SetHwCmpValue1 (uint32_t *baseAddr*, uint16_t *value*) [inline], [static]

This function sets the compare value of the lower limitation for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>value</i>	Setting value.

3.2.3.15 static void ADC16_HAL_SetHwCmpValue2 (uint32_t *baseAddr*, uint16_t *value*) [inline], [static]

This function sets the compare value of the higher limitation for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>value</i>	Setting value.

3.2.3.16 static bool ADC16_HAL_GetConvActiveCmd (uint32_t *baseAddr*) [inline], [static]

This function checks whether the converter is active for the ADC module. If it is dis-asserted when the conversion is completed, one of the completed flag is asserted for the indicated group mux. See the "ADC16_HAL_GetChnConvCompletedCmd()".

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

Assertion of that the converter is active.

3.2.3.17 static void ADC16_HAL_SetHwTriggerCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the hardware trigger mode for the ADC module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.18 static void ADC16_HAL_SetHwCmpCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the hardware comparator for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

ADC16 HAL Driver

<i>enable</i>	Switcher to asserted the feature.
---------------	-----------------------------------

3.2.3.19 static void ADC16_HAL_SetHwCmpGreaterCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the setting that is greater than the hardware comparator.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.20 static void ADC16_HAL_SetHwCmpRangeCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the setting of range for the hardware comparator.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.2.3.21 void ADC16_HAL_SetHwCmpMode (*uint32_t baseAddr, adc16_hw_cmp_range_mode_t mode*)

This function configures the asserted range of the hardware comparator for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of range mode, see to "adc16_hw_cmp_range_mode_t".

3.2.3.22 static void ADC16_HAL_SetRefVoltSrcMode (*uint32_t baseAddr, adc16_ref_volt_src_mode_t mode*) [inline], [static]

This function selects the reference voltage source for the ADC16 module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of asserted the feature.

3.2.3.23 static void ADC16_HAL_SetContinuousConvCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the continuous conversion mode for the ADC module. Once enabled, continuous conversions, or sets of conversions if the hardware average function, is enabled after initiating a conversion.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to asserted the feature.

3.3 ADC16 Peripheral Driver

3.3.1 Overview

This section describes the programming interface of the ADC16 Peripheral driver. The ADC16 peripheral driver configures the ADC16 (16-bit SAR Analog-to-Digital Converter). It handles calibration, initialization, and configuration of 16-bit SAR ADC module.

3.3.2 ADC16 Driver model building

ADC16 driver has three parts:

- Basic Converter - This part handles the mechanism that converts the external analog voltage to a digital value. API functions configure the converter.
- Channel Mux - Multiple channels share the converter in each ADC instance because of the time division multiplexing. However, the converter can only handle one channel at a time. To get the value of an indicated channel, the channel mux should be set to the connection between an indicated pad and the converter's input. The conversion value during this period is for the channel only. API functions configure the channel.
- Advance Feature Group - The advanced feature group covers optional features for applications. These features includes some that are already implemented by hardware, such as the calibration, hardware average, hardware compare, different power, and speed mode. APIs configure the advanced features. Although these features are optional, they are recommended to ensure that the ADC performs better, especially for calibration.

3.3.3 ADC16 Initialization

Note that the calibration should be done before any other operation.

To initialize the ADC16 driver, prepare a configuration structure and populate it with an available configuration. API functions are designed for typical use cases and facilitate populating the structure. See the "Call diagram" section for typical use cases. Additionally, the application should provide a block of memory to keep the state while the driver operates. After the configuration structure is available and memory is allocated to keep state, the ADC module can be initialized by calling the API of [ADC16_DRV_Init\(\)](#) function.

3.3.4 ADC16 Call diagram

There are three kinds of typical use cases for the ADC module:

- Interrupt mode - Interrupt mode works only with continuous conversion or one-time conversion. To use the interrupt mode, the user enables the interrupt when configuring for each conversion. Note that the conversion value should be read out by calling the API of the "ADC16_DRV_GetConvValueRAW()" in the user-defined ISR. Use the interrupt mode carefully with the continuous

conversion because too many interrupts may affect the main routine. Using the low conversion speed is recommended in the continuous interrupt mode.

- Blocking mode. Blocking mode is based on polling mode and works only with the continuous conversion mode. The conversion is triggered by the channel configuration. When the conversion is completed, it gets blocked because the result data is not read. The application obtains the result data by calling API. After this, the conversion becomes continuous.
- One-Time-Trigger Mode. One-Time-Trigger works without an interrupt and continuous mode. After it is triggered by channel configuration, the conversion launches and the application gets the result data. There is no auto operation to make the conversion continuous. The application should trigger the conversion again if another conversion is needed.

These use cases are based on the software trigger. However, they can easily be ported to use the hardware trigger. If using the hardware trigger, enable it when initializing the converter.

Complex use cases, such as the DMA and the multiple channel scan, require another module to work correctly. They can be customized according to the application.

These are the examples to initialize and configure the ADC driver for typical use cases.

Common files:

adc16_test.h

```
// adc16_test.h //

#ifndef __ADC16_TEST_H__
#define __ADC16_TEST_H__


#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include "board.h"
#include "fsl_os_abstraction.h"
#include "fsl_debug_console.h"

#include "fsl_adc16_driver.h"

#define ADC16_TEST_CHN_TEMPERATURE      (26U) // Temperature Sensor. //
#define ADC16_TEST_CHN_GROUT_NUM        (0U)

#define ADC16_TEST_LOOP_CYCLE          (4U)

void ADC16_TEST_InstallCallback(uint32_t instance, void (*callbackFunc) (void) );

void ADC16_TEST_OneTimeTrigger(uint32_t instance, uint32_t chnGroup, uint8_t chn);
void ADC16_TEST_Blocking(uint32_t instance, uint32_t chnGroup, uint8_t chn);
void ADC16_TEST_IntMode(uint32_t instance, uint32_t chnGroup, uint8_t chn);
void ADC16_TEST_AdvancedFeature(uint32_t instance, uint32_t chnGroup, uint8_t chn);

#endif // __ADC16_TEST_H__ //
```

fsl_adc16_irq.c

```
#include <stdint.h>
#include <stdbool.h>
#include "fsl_adc16_driver.h"

// Define array to keep run-time callback set by application. //
```

ADC16 Peripheral Driver

```
void (* volatile g_AdctestCallback[HW_ADC_INSTANCE_COUNT])(void);

void ADC16_TEST_InstallCallback(uint32_t instance, void (*callbackFunc)(void) )
{
    g_AdctestCallback[instance] = callbackFunc;
}

static void ADC16_TEST_IRQHandler(uint32_t instance)
{
    if (g_AdctestCallback[instance])
    {
        (void) (* (g_AdctestCallback[instance]))();
    }
}

// ADC IRQ handler that would cover the same name's APIs in startup code //
void ADC0_IRQHandler(void)
{
    // Add user-defined ISR for ADC0. //
    ADC16_TEST_IRQHandler(0U);
}

#if (HW_ADC_INSTANCE_COUNT > 1U)
void ADC1_IRQHandler(void)
{
    // Add user-defined ISR for ADC1. //
    ADC16_TEST_IRQHandler(1U);
}
#endif

#if (HW_ADC_INSTANCE_COUNT > 2U)
void ADC2_IRQHandler(void)
{
    // Add user-defined ISR for ADC2. //
    ADC16_TEST_IRQHandler(2U);
}
#endif

#if (HW_ADC_INSTANCE_COUNT > 3U)
void ADC3_IRQHandler(void)
{
    // Add user-defined ISR for ADC3. //
    ADC16_TEST_IRQHandler(3U);
}
#endif
```

Interrupt Mode:

```
// adc16_test_int.c //

#include "adc16_test.h"

static volatile bool      g_AdcConvIntCompleted = false;
static volatile uint16_t   g_adcValueInt = 0U;
static volatile uint32_t   g_curInstance = 0U;
static volatile uint32_t   g_curChnGroup = 0U;
void ADC16_TEST_MyISR(void);

void ADC16_TEST_IntMode(uint32_t instance, uint32_t chnGroup, uint8_t chn)
{
#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t MyAdcCalibrationParam;
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_user_config_t MyAdcUserConfig;
    adc16_chn_config_t MyChnConfig;
    uint32_t i;
```

```

#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    // Auto calibration. //
    ADC16_DRV_GetAutoCalibrationParam(instance, &MyAdcCalibraitionParam);
    ADC16_DRV_SetCalibrationParam(instance, &MyAdcCalibraitionParam);
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //

    // Initialization for interrupt mode. //
    ADC16_DRV_StructInitUserConfigDefault(&MyAdcUserConfig);
    MyAdcUserConfig.intEnable = true; // Enable interrupt for ADC conversion completed. //
    ADC16_DRV_Init(instance, &MyAdcUserConfig);

    // Install Callback function into ISR. //
    g_curInstance = instance;
    g_curChnGroup = chnGroup;
    ADC_TEST_InstallCallback(instance, ADC_TEST_MyISR);
    // Trigger indicated channel. //
    MyChnConfig.chnNum = chn;
    MyChnConfig.diffEnable = false;
    MyChnConfig.intEnable = true;
    MyChnConfig.chnMux = kAdcChnMuxOfA;

    for (i = 0U; i < 4U; i++)
    {
        // Start the conversion. //
        ADC16_DRV_ConfigConvChn(instance, chnGroup, &MyChnConfig);

        // Wait the interrupt for conversion completed. //
        while (!g_AdcConvIntCompleted) {}

        // Print the conversion result. //
        printf("ADC16_DRV_GetConvValueRAW: 0x%X\t", g_adcValueInt);
        printf("ADC16_DRV_ConvRAWData: %ld\r\n",
               ADC16_DRV_ConvRAWData(g_adcValueInt, false,
                                     kAdcResolutionBitOfSingleEndAs12) );
        g_AdcConvIntCompleted = false;
    }

    // Pause the conversion after testing. //
    ADC16_DRV_PauseConv(instance, chnGroup);

    // Disable the ADC. //
    ADC16_DRV_Deinit(instance);
}

void ADC16_TEST_MyISR(void)
{
    g_adcValueInt = ADC16_DRV_GetConvValueRAW(g_curInstance, g_curChnGroup);
    g_AdcConvIntCompleted = true;
}

```

Blocking Mode:

```

// adc16_test_blocking.c //

#include "adc16_test.h"

void ADC16_TEST_Blocking(uint32_t instance, uint32_t chnGroup, uint8_t chn)
{
#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t MyAdcCalibraitionParam;
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //
    adc16_user_config_t MyAdcUserConfig;
    adc16_chn_config_t MyChnConfig;
    uint16_t MyAdcValue;

```

ADC16 Peripheral Driver

```
uint32_t i;

#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    // Auto calibration. //
    ADC16_DRV_GetAutoCalibrationParam(instance, &MyAdcCalibrationParam);
    ADC16_DRV_SetCalibrationParam(instance, &MyAdcCalibrationParam);
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //

    // Initialize the ADC converter. //
    ADC16_DRV_StructInitUserConfigDefault(&MyAdcUserConfig);
    MyAdcUserConfig.continuousConvEnable = true; // Enable continuous conversion. //
    ADC16_DRV_Init(instance, &MyAdcUserConfig);

    // Configure the ADC channel and take an initial trigger. //
    MyChnConfig.chnNum = chn;
    MyChnConfig.diffEnable= false;
    MyChnConfig.intEnable = false;
    MyChnConfig.chnMux = kAdcChnMuxOfA;
    ADC16_DRV_ConfigConvChn(instance, chnGroup, &MyChnConfig);

    for (i = 0U; i < 4U; i++)
    {
        // Wait for the most recent conversion.//
        ADC16_DRV_WaitConvDone(instance, chnGroup);

        // Fetch the conversion value and format it. //
        MyAdcValue = ADC16_DRV_GetConvValueRAW(instance, chnGroup);
        printf("ADC16_DRV_GetConvValueRAW: 0x%X\t", MyAdcValue);
        printf("ADC16_DRV_ConvRAWData: %ld\r\n",
            ADC16_DRV_ConvRAWData(MyAdcValue, false,
            kAdcResolutionBitOfSingleEndAs12) );
    }

    // Pause the conversion after testing. //
    ADC16_DRV_PauseConv(instance, chnGroup);
    // Disable the ADC. //
    ADC16_DRV_Deinit(instance);
}
```

One-Time-Trigger Mode:

```
// adc16_test_one_time_trigger.c //

#include "adc16_test.h"

void ADC16_TEST_OneTimeTrigger(uint32_t instance, uint32_t chnGroup, uint8_t chn)
{
#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t MyAdcCalibrationParam;
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //
    adc16_user_config_t MyAdcUserConfig;
    adc16_chn_config_t MyChnConfig;
    int32_t MyAdcValue;
    uint32_t i;

#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    // Auto calibration. //
    ADC16_DRV_GetAutoCalibrationParam(instance, &MyAdcCalibrationParam);
    ADC16_DRV_SetCalibrationParam(instance, &MyAdcCalibrationParam);
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //

    // Initialize the ADC converter. //
    ADC16_DRV_StructInitUserConfigDefault(&MyAdcUserConfig);
    ADC16_DRV_Init(instance, &MyAdcUserConfig);

    // Configuration for ADC channel. //
```

```

MyChnConfig.chnNum = chn;
MyChnConfig.diffEnable= false;
MyChnConfig.intEnable = false;
#ifndef FSL_FEATURE_ADC16_HAS_MUX_SELECT
    MyChnConfig.chnMux = kAdcChnMuxOfA;
#endif // FSL_FEATURE_ADC16_HAS_MUX_SELECT //

for (i = 0U; i < 4U; i++)
{
    // Trigger the conversion with indicated channel's configuration. //
    ADC16_DRV_ConfigConvChn(instance, chnGroup, &MyChnConfig);

    // Wait for the conversion to be done. //
    ADC16_DRV_WaitConvDone(instance, chnGroup);

    // Fetch the conversion value and format it. //
    MyAdcValue = ADC16_DRV_GetConvValueRAW(instance, chnGroup);
    printf("ADC16_DRV_GetConvValueRAW: 0x%X\t", MyAdcValue);
    printf("ADC16_DRV_ConvRAWData: %ld\r\n",
          ADC16_DRV_ConvRAWData(MyAdcValue, false,
          kAdcResolutionBitOfSingleEndAs12) );
}
// Pause the conversion after testing. //
ADC16_DRV_PauseConv(instance, chnGroup);
// Disable the ADC. //
ADC16_DRV_Deinit(instance);
}

```

For Advanced Features:

```

// adc16_test_advanced_feature.c //

#include "adc16_test.h"

void ADC16_TEST_AdvancedFeature(uint32_t instance, uint32_t chnGroup, uint8_t chn)
{
#ifndef FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t MyAdcCalibraitionParam;
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //
    adc16_user_config_t MyAdcUserConfig;
    adc16_chn_config_t MyChnConfig;
    uint16_t MyAdcValue;
    uint32_t i;
    adc16_hw_cmp_config_t MyAdcHwCmpConfig;

#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    // Auto calibraion. //
    ADC16_DRV_GetAutoCalibrationParam(instance, &MyAdcCalibraitionParam);
    ADC16_DRV_SetCalibrationParam(instance, &MyAdcCalibraitionParam);
#endif // FSL_FEATURE_ADC16_HAS_CALIBRATION //

    // Initialization for interrupt mode. //
    ADC16_DRV_StructInitUserConfigDefault(&MyAdcUserConfig);
    ADC16_DRV_Init(instance, &MyAdcUserConfig);

    // Setting for advanced feature. //

    // Hardware compare. //
    // Available range is between cmpValue1 and cmpValue2. //
    MyAdcHwCmpConfig.cmpRangeMode = kAdcHwCmpRangeModeOf4;
    MyAdcHwCmpConfig.cmpValue1 = 0x0010U;
    MyAdcHwCmpConfig.cmpValue2 = 0xFFFF0U;
    ADC16_DRV_EnableHwCmp(instance, &MyAdcHwCmpConfig);

    // Long sample mode. //
    ADC16_DRV_EnableLongSample(instance,

```

ADC16 Peripheral Driver

```
kAdcLongSampleCycleOf16);

// Hardware average. //
#if FSL_FEATURE_ADC16_HAS_HW_AVERAGE
    ADC16_DRV_EnableHwAverage(instance, kAdcHwAverageCountOf32);
#endif // FSL_FEATURE_ADC16_HAS_HW_AVERAGE //

// Setting channel as one time trigger. //
MyChnConfig.chnNum = chn;
MyChnConfig.diffEnable= false;
MyChnConfig.intEnable = false;
MyChnConfig.chnMux = kAdcChnMuxOfA;

for (i = 0U; i < ADC_TEST_LOOP_CYCLE; i++)
{
    // Trigger the conversion with indicated channel's configuration. //
    ADC16_DRV_ConfigConvChn(instance, chnGroup, &MyChnConfig);

    // Wait for the conversion to be done. //
    ADC16_DRV_WaitConvDone(instance, chnGroup);

    // Fetch the conversion value and format it. //
    MyAdcValue = ADC16_DRV_GetConvValueRAW(instance, chnGroup);
    printf("ADC16_DRV_GetConvValueRAW: 0x%X\t", MyAdcValue);
    printf("ADC16_DRV_ConvRAWData: %ld\r\n",
        ADC16_DRV_ConvRAWData(MyAdcValue, false,
        kAdcResolutionBitOfSingleEndAs12) );
}

// Pause the conversion after testing. //
ADC16_DRV_PauseConv(instance, chnGroup);

// Disable the ADC. //
ADC16_DRV_Deinit(instance);
}
```

Data Structures

- struct [adc16_hw_cmp_config_t](#)
Defines the structure to configure the ADC16 module hardware compare. [More...](#)
- struct [adc16_chn_config_t](#)
Defines the structure to configure the ADC16 channel. [More...](#)
- struct [adc16_user_config_t](#)
Defines the structure to initialize the ADC16 module converter. [More...](#)

Enumerations

- enum [adc16_flag_t](#) {
 kAdcConvActiveFlag = 0U,
 kAdcChnConvCompleteFlag = 3U }
Defines the type of event flags.

Functions

- [adc16_status_t ADC16_DRV_StructInitUserConfigDefault](#) ([adc16_user_config_t](#) *userConfigPtr)

- `adc16_status_t ADC16_DRV_Init` (uint32_t instance, `adc16_user_config_t` *userConfigPtr)
 - Fills the initial user configuration by default for a one-time trigger mode.*
 - Initializes the ADC module converter.*
- `void ADC16_DRV_Deinit` (uint32_t instance)
 - De-initializes the ADC module converter.*
- `void ADC16_DRV_EnableLongSample` (uint32_t instance, `adc16_long_sample_cycle_mode_t` mode)
 - Enables the long sample mode feature.*
- `void ADC16_DRV_DisableLongSample` (uint32_t instance)
 - Disables the long sample mode feature.*
- `adc16_status_t ADC16_DRV_EnableHwCmp` (uint32_t instance, `adc16_hw_cmp_config_t` *hwCmpConfigPtr)
 - Enables the hardware compare feature.*
- `void ADC16_DRV_DisableHwCmp` (uint32_t instance)
 - Disables the hardware compare feature.*
- `adc16_status_t ADC16_DRV_ConfigConvChn` (uint32_t instance, uint32_t chnGroup, `adc16_chn_config_t` *chnConfigPtr)
 - Configures the conversion channel by software.*
- `void ADC16_DRV_WaitConvDone` (uint32_t instance, uint32_t chnGroup)
 - Waits for the latest conversion to be complete.*
- `void ADC16_DRV_PauseConv` (uint32_t instance, uint32_t chnGroup)
 - Pauses the current conversion by software.*
- `uint16_t ADC16_DRV_GetConvValueRAW` (uint32_t instance, uint32_t chnGroup)
 - Gets the latest conversion value.*
- `int32_t ADC16_DRV_ConvRAWData` (uint16_t convValue, bool diffEnable, `adc16_resolution_mode_t` mode)
 - Formats the initial value fetched from the ADC16 module.*
- `bool ADC16_DRV_GetFlag` (uint32_t instance, `adc16_flag_t` flag)
 - Gets the event status of the ADC16 module.*
- `bool ADC16_DRV_GetChnFlag` (uint32_t instance, uint32_t chnGroup, `adc16_flag_t` flag)
 - Gets the event status of each channel group.*

Variables

- `const uint32_t g_adcBaseAddr []`
 - Table of base addresses for ADC16 instances.*
- `const IRQn_Type g_adcIrqId [HW_ADC_INSTANCE_COUNT]`
 - Table to save ADC IRQ enum numbers defined in the CMSIS header file.*

3.3.5 Data Structure Documentation

3.3.5.1 struct adc16_hw_cmp_config_t

This structure holds the configuration for the ADC16 internal comparator.

ADC16 Peripheral Driver

Data Fields

- `uint16_t cmpValue1`
Value for CMP value 1.
- `uint16_t cmpValue2`
Value for CMP value 2.

3.3.5.1.0.1 Field Documentation

3.3.5.1.0.1.1 `uint16_t adc16_hw_cmp_config_t::cmpValue1`

3.3.5.1.0.1.2 `uint16_t adc16_hw_cmp_config_t::cmpValue2`

Select the available range to pass the comparator.

3.3.5.2 `struct adc16_chn_config_t`

This structure holds the ADC16 channel configuration.

Data Fields

- `uint32_t chnNum`
Selection of input channel number.
- `bool intEnable`
Enable trigger interrupt when the conversion is complete.
- `bool diffEnable`
Enable setting the differential mode for conversion.

3.3.5.2.0.2 Field Documentation

3.3.5.2.0.2.1 `uint32_t adc16_chn_config_t::chnNum`

3.3.5.2.0.2.2 `bool adc16_chn_config_t::intEnable`

3.3.5.2.0.2.3 `bool adc16_chn_config_t::diffEnable`

3.3.5.3 `struct adc16_user_config_t`

This structure holds the configuration for the ADC16 module converter.

Data Fields

- `bool intEnable`
Enable internal ISR.
- `bool lowPowerEnable`
Enable low power mode for converter.
- `adc16_clk_divider_mode_t clkDividerMode`
Select divider of input clock for converter.

- **adc16_resolution_mode_t resolutionMode**
Select conversion resolution for converter.
- **adc16_clk_src_mode_t clkSrcMode**
Select source of input clock for converter.
- **bool asyncClkEnable**
Enable asynchronous clock output at when initializing converter.
- **bool highSpeedEnable**
Enable the high speed mode for converter.
- **bool hwTriggerEnable**
Enable triggering by hardware.
- **adc16_ref_volt_src_mode_t refVoltSrcMode**
Select the reference voltage source for converter.
- **bool continuousConvEnable**
Enable launching the continuous conversion mode.

3.3.5.3.0.3 Field Documentation

3.3.5.3.0.3.1 **bool adc16_user_config_t::intEnable**

3.3.5.3.0.3.2 **bool adc16_user_config_t::lowPowerEnable**

3.3.5.3.0.3.3 **adc16_clk_divider_mode_t adc16_user_config_t::clkDividerMode**

3.3.5.3.0.3.4 **adc16_resolution_mode_t adc16_user_config_t::resolutionMode**

3.3.5.3.0.3.5 **adc16_clk_src_mode_t adc16_user_config_t::clkSrcMode**

3.3.5.3.0.3.6 **bool adc16_user_config_t::asyncClkEnable**

3.3.5.3.0.3.7 **bool adc16_user_config_t::highSpeedEnable**

3.3.5.3.0.3.8 **bool adc16_user_config_t::hwTriggerEnable**

3.3.5.3.0.3.9 **adc16_ref_volt_src_mode_t adc16_user_config_t::refVoltSrcMode**

3.3.5.3.0.3.10 **bool adc16_user_config_t::continuousConvEnable**

3.3.6 Enumeration Type Documentation

3.3.6.1 **enum adc16_flag_t**

Enumerator

kAdcConvActiveFlag Indicates if a conversion or hardware averaging is in progress.

kAdcChnConvCompleteFlag Indicates if the channel group A is ready.

3.3.7 Function Documentation

3.3.7.1 `adc16_status_t ADC16_DRV_StructInitUserConfigDefault (adc16_user_config_t * userConfigPtr)`

This function fills the initial user configuration by default for a one-time trigger mode. Calling the initialization function with the filled parameter configures the ADC module work as one-time trigger mode. The settings are:

- .intEnable = false;
- .lowPowerEnable = false;
- .clkDividerMode = kAdcClkDividerInputOf8;
- .resolutionMode = kAdcResolutionBitOf12or13;
- .clkSrcMode = kAdcClkSrcOfBusClk;
- .asyncClkEnable = false;
- .highSpeedEnable = false;
- .hwTriggerEnable = false;
- .dmaEnable = false;
- .refVoltSrcMode = kAdcRefVoltSrcOfVref;
- .continuousConvEnable = false;

Parameters

<i>userConfigPtr</i>	Pointer to the user configuration structure. See the "adc16_user_config_t".
----------------------	---

Returns

Execution status.

3.3.7.2 `adc16_status_t ADC16_DRV_Init (uint32_t instance, adc16_user_config_t * userConfigPtr)`

This function initializes the converter in the ADC module. Regardless of the completed calibration for the device, this API function with initial configuration, should be called before any other operations. Initial configurations are mainly for the converter itself. For advanced features, the corresponding APIs should be called after this function.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>userConfigPtr</i>	Pointer to the initialization structure. See the "adc16_user_config_t".

Returns

Execution status.

3.3.7.3 void ADC16_DRV_Deinit (uint32_t *instance*)

This function de-initializes and gates the ADC module. When ADC is no longer used, calling this API function shuts down the device to reduce the power consumption.

Parameters

<i>instance</i>	ADC16 instance ID.
-----------------	--------------------

3.3.7.4 void ADC16_DRV_EnableLongSample (uint32_t *instance*, adc16_long_sample_cycle_mode_t *mode*)

This function enables the long sample mode feature and sets it with the indicated configuration. Launching the long sample mode adjusts the sample period allowing accurate sampling of the higher impedance inputs.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>mode</i>	Selection of configuration, see to "adc16_long_sample_cycle_mode_t".

3.3.7.5 void ADC16_DRV_DisableLongSample (uint32_t *instance*)

This API function disables the long sample mode feature.

Parameters

<i>instance</i>	ADC16 instance ID.
-----------------	--------------------

3.3.7.6 adc16_status_t ADC16_DRV_EnableHwCmp (uint32_t *instance*, adc16_hw_cmp_config_t * *hwCmpConfigPtr*)

This API enables the hardware compare feature with the indicated configuration. Launching the hardware compare ensures that the conversion, which results in a predefined range, can only be accepted. Values

ADC16 Peripheral Driver

out of range are ignored during conversion.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>hwCmpConfigPtr</i>	Pointer to a configuration structure. See the "adc16_hw_cmp_config_t".

Returns

Execution status.

3.3.7.7 void ADC16_DRV_DisableHwCmp (uint32_t *instance*)

This API function disables the hardware compare feature.

Parameters

<i>instance</i>	ADC16 instance ID.
-----------------	--------------------

3.3.7.8 adc16_status_t ADC16_DRV_ConfigConvChn (uint32_t *instance*, uint32_t *chnGroup*, adc16_chn_config_t * *chnConfigPtr*)

This function configures the conversion channel. When the ADC16 module has been initialized by enabling the software trigger (disable hardware trigger), calling this API triggers the conversion.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>chnGroup</i>	Selection of the configuration group.
<i>chnConfigPtr</i>	Pointer to the configuration structure. See the "adc16_chn_config_t".

Returns

Execution status.

3.3.7.9 void ADC16_DRV_WaitConvDone (uint32_t *instance*, uint32_t *chnGroup*)

This function waits for the latest conversion to be complete. When triggering the conversion by configuring the available channel, the converter is launched. This API function should be called to wait for the conversion to be complete when no interrupt or DMA mode is used for the ADC16 module. After the waiting period, the available data from the conversion result are fetched. The complete flag is not cleared until the result data is read.

ADC16 Peripheral Driver

Parameters

<i>instance</i>	ADC16 instance ID.
<i>chnGroup</i>	Selection of configuration group.

Returns

Execution status.

3.3.7.10 void ADC16_DRV_PauseConv (uint32_t *instance*, uint32_t *chnGroup*)

This function pauses the current conversion setting by software.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>chnGroup</i>	Selection of configuration group.

3.3.7.11 uint16_t ADC16_DRV_GetConvValueRAW (uint32_t *instance*, uint32_t *chnGroup*)

This function gets the conversion value from the ADC16 module.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>chnGroup</i>	Selection of configuration group.

Returns

Unformatted conversion value.

3.3.7.12 int32_t ADC16_DRV_ConvRAWData (uint16_t *convValue*, bool *diffEnable*, adc16_resolution_mode_t *mode*)

This function formats the initial value fetched from the ADC16 module. Initial value fetched from the ADC module can't be read as a number, especially for the signed value generated by the differential conversion. This function can format the initial value to be a readable one.

Parameters

<i>convValue</i>	Initial value directly from the register.
<i>diffEnable</i>	Identifier for the differential mode.
<i>mode</i>	Formatted data resolution mode.

Returns

Formatted conversion value.

3.3.7.13 **bool ADC16_DRV_GetFlag (uint32_t *instance*, adc16_flag_t *flag*)**

This function gets the event status of the ADC16 converter. If the event is asserted, it returns "true". Otherwise, it is "false".

Parameters

<i>instance</i>	ADC16 instance ID.
<i>flag</i>	Indicated event.

Returns

Assertion of event flag.

3.3.7.14 **bool ADC16_DRV_GetChnFlag (uint32_t *instance*, uint32_t *chnGroup*, adc16_flag_t *flag*)**

This function gets the event status of each channel group. If the event is asserted, it returns "true". Otherwise, it is "false".

Parameters

<i>instance</i>	ADC16 instance ID.
<i>chnGroup</i>	ADC16 channel group number.
<i>flag</i>	Indicated event.

Returns

Assertion of event flag.

3.3.8 Variable Documentation

3.3.8.1 `const uint32_t g_adcBaseAddr[]`

3.3.8.2 `const IRQn_Type g_adcIrqId[HW_ADC_INSTANCE_COUNT]`

Chapter 4

Comparator (CMP)

4.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Comparator (CMP) block of Kinetis devices.

Modules

- [CMP HAL Driver](#)
- [CMP Peripheral Driver](#)

4.2 CMP HAL Driver

4.2.1 Overview

This section describes the programming interface of the CMP HAL driver.

Enumerations

- enum `cmp_status_t` {
 `kStatus_CMP_Success` = 0U,
 `kStatus_CMP_InvalidArgument` = 1U,
 `kStatus_CMP_Failed` = 2U }
 CMP status return codes.
- enum `cmp_hysteresis_mode_t` {
 `kCmpHysteresisOfLevel0` = 0U,
 `kCmpHysteresisOfLevel1` = 1U,
 `kCmpHysteresisOfLevel2` = 2U,
 `kCmpHysteresisOfLevel3` = 3U }
 Define the selections of hard block hysteresis control level.
- enum `cmp_filter_counter_mode_t` {
 `kCmpFilterCountSampleOf0` = 0U,
 `kCmpFilterCountSampleOf1` = 1U,
 `kCmpFilterCountSampleOf2` = 2U,
 `kCmpFilterCountSampleOf3` = 3U,
 `kCmpFilterCountSampleOf4` = 4U,
 `kCmpFilterCountSampleOf5` = 5U,
 `kCmpFilterCountSampleOf6` = 6U,
 `kCmpFilterCountSampleOf7` = 7U }
 Define the selections of filter sample counter.
- enum `cmp_dac_ref_volt_src_mode_t` {
 `kCmpDacRefVoltSrcOf1` = 0U,
 `kCmpDacRefVoltSrcOf2` = 1U }
 Define the selections of reference voltage source for internal DAC.
- enum `cmp_chn_mux_mode_t` {
 `kCmpInputChn0` = 0U,
 `kCmpInputChn1` = 1U,
 `kCmpInputChn2` = 2U,
 `kCmpInputChn3` = 3U,
 `kCmpInputChn4` = 4U,
 `kCmpInputChn5` = 5U,
 `kCmpInputChn6` = 6U,
 `kCmpInputChn7` = 7U,
 `kCmpInputChnDac` = `kCmpInputChn7` }
 Define the selection of CMP channel mux.

Functions

- void **CMP_HAL_Init** (uint32_t baseAddr)

Reset CMP's registers to a known state.
- static void **CMP_HAL_SetFilterCounterMode** (uint32_t baseAddr, **cmp_filter_counter_mode_t** mode)

Set the filter sample count.
- static void **CMP_HAL_SetHystersisMode** (uint32_t baseAddr, **cmp_hystersis_mode_t** mode)

Set the programmable hystersis level.
- static void **CMP_HAL_Enable** (uint32_t baseAddr)

Enable the comparator in CMP module.
- static void **CMP_HAL_Disable** (uint32_t baseAddr)

Disable the comparator in CMP module.
- static void **CMP_HAL_SetOutputPinCmd** (uint32_t baseAddr, bool enable)

Switch to enable the compare output signal connecting to pin.
- static void **CMP_HAL_SetUnfilteredOutCmd** (uint32_t baseAddr, bool enable)

Switch to enable the filter for output of compare logic in CMP module.
- static void **CMP_HAL_SetInvertLogicCmd** (uint32_t baseAddr, bool enable)

Switch to enable the polarity of the analog comparator function.
- static void **CMP_HAL_SetHighSpeedCmd** (uint32_t baseAddr, bool enable)

Switch to enable the power (speed) comparison mode in CMP module.
- static void **CMP_HAL_SetSampleModeCmd** (uint32_t baseAddr, bool enable)

Switch to enable the sample mode in CMP module.
- static void **CMP_HAL_SetFilterPeriodValue** (uint32_t baseAddr, uint8_t value)

Set the filter sample period in CMP module.
- static bool **CMP_HAL_GetOutputLogic** (uint32_t baseAddr)

Get the comparator logic output in CMP module.
- static bool **CMP_HAL_GetOutputFallingFlag** (uint32_t baseAddr)

Get the logic output's falling edge event in CMP module.
- static void **CMP_HAL_ClearOutputFallingFlag** (uint32_t baseAddr)

Clear the logic output's falling edge event in CMP module.
- static bool **CMP_HAL_GetOutputRisingFlag** (uint32_t baseAddr)

Get the logic output's rising edge event in CMP module.
- static void **CMP_HAL_ClearOutputRisingFlag** (uint32_t baseAddr)

Clear the logic output's rising edge event in CMP module.
- static void **CMP_HAL_SetOutputFallingIntCmd** (uint32_t baseAddr, bool enable)

Switch to enable requesting interrupt when falling-edge on COUT has occurred.
- static bool **CMP_HAL_GetOutputFallingIntCmd** (uint32_t baseAddr)

Get the switcher of interrupt request on COUT's falling-edge.
- static void **CMP_HAL_SetOutputRisingIntCmd** (uint32_t baseAddr, bool enable)

Get the switcher of interrupt request on COUT's rising-edge.
- static bool **CMP_HAL_GetOutputRisingIntCmd** (uint32_t baseAddr)

Get the switcher of interrupt request on COUT's rising-edge.
- static void **CMP_HAL_SetDacCmd** (uint32_t baseAddr, bool enable)

Switch to enable internal 6-bit DAC in CMP module.
- static void **CMP_HAL_SetDacRefVoltSrcMode** (uint32_t baseAddr, **cmp_dac_ref_volt_src_mode_t** mode)

Set the reference voltage source for internal 6-bit DAC in CMP module.
- static void **CMP_HAL_SetDacValue** (uint32_t baseAddr, uint8_t value)

Set the output value for internal 6-bit DAC in CMP module.
- static void **CMP_HAL_SetPlusInputChnMuxMode** (uint32_t baseAddr, **cmp_chn_mux_mode_t** mode)

Set the plus input channel multiplexer mode.

t mode)

Set the plus channel for analog comparator.

- static void [CMP_HAL_SetMinusInputChnMuxMode](#) (uint32_t baseAddr, [cmp_chn_mux_mode_t](#) mode)

Set the minus channel for analog comparator.

4.2.2 Enumeration Type Documentation

4.2.2.1 enum cmp_status_t

Enumerator

kStatus_CMP_Success Success.

kStatus_CMP_InvalidArgument Invalid argument existed.

kStatus_CMP_Failed Execution failed.

4.2.2.2 enum cmp_hysteresis_mode_t

The hysteresis control level indicates the smallest window between the two input when asserting the change of output. it should be referred to the Data Sheet for detail electrical characteristics. Generally, the lower level represents the smaller window.

Enumerator

kCmpHysteresisOfLevel0 Level 0.

kCmpHysteresisOfLevel1 Level 1.

kCmpHysteresisOfLevel2 Level 2.

kCmpHysteresisOfLevel3 Level 3.

4.2.2.3 enum cmp_filter_counter_mode_t

The selection item represents the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state.

Enumerator

kCmpFilterCountSampleOf0 Disable the filter.

kCmpFilterCountSampleOf1 One sample must agree.

kCmpFilterCountSampleOf2 2 consecutive samples must agree

kCmpFilterCountSampleOf3 3 consecutive samples must agree

kCmpFilterCountSampleOf4 4 consecutive samples must agree

kCmpFilterCountSampleOf5 5 consecutive samples must agree

kCmpFilterCountSampleOf6 6 consecutive samples must agree

kCmpFilterCountSampleOf7 7 consecutive samples must agree

4.2.2.4 enum cmp_dac_ref_volt_src_mode_t

Enumerator

kCmpDacRefVoltSrcOf1 Vin1 - Vref_out.

kCmpDacRefVoltSrcOf2 Vin2 - Vdd.

4.2.2.5 enum cmp_chn_mux_mode_t

Enumerator

kCmpInputChn0 Comparator input channel 0.

kCmpInputChn1 Comparator input channel 1.

kCmpInputChn2 Comparator input channel 2.

kCmpInputChn3 Comparator input channel 3.

kCmpInputChn4 Comparator input channel 4.

kCmpInputChn5 Comparator input channel 5.

kCmpInputChn6 Comparator input channel 6.

kCmpInputChn7 Comparator input channel 7.

kCmpInputChnDac Comparator input channel 7.

4.2.3 Function Documentation

4.2.3.1 void CMP_HAL_Init(uint32_t baseAddr)

This function is to reset CMP's registers to a known state. This state is defined in Reference Manual, which is power on reset value.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

4.2.3.2 static void CMP_HAL_SetFilterCounterMode(uint32_t baseAddr, cmp_filter_counter_mode_t mode) [inline], [static]

This function is to set the filter sample count. The value of filter sample count represents the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state.

CMP HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Filter count value mode, see to "cmp_filter_counter_mode_t".

4.2.3.3 static void CMP_HAL_SetHysteresisMode (uint32_t *baseAddr*, cmp_hysteresis_mode_t *mode*) [inline], [static]

This function is to define the programmable hysteresis level. The hysteresis values associated with each level are device-specific. See the Data Sheet of the device for the exact values. Also, see to "cmp_hysteresis_mode_t" for some additional information.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Hysteresis level, see to "cmp_hysteresis_mode_t".

4.2.3.4 static void CMP_HAL_Enable (uint32_t *baseAddr*) [inline], [static]

This function is to enable the comparator in CMP module. The analog comparator is the core component in CMP module. Only when it is enabled, all the other functions for advanced features are meaningful.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

4.2.3.5 static void CMP_HAL_Disable (uint32_t *baseAddr*) [inline], [static]

This function is to Disable the comparator in CMP module. The analog comparator is the core component in CMP module. When the it is disabled, it remains in the off state, and consumes no power.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

4.2.3.6 static void CMP_HAL_SetOutputPinCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function is to switch to enable the compare output signal connecting to pin. The comparator output (CMPO) is driven out on the associated CMPO output pin if the comparator owns the pin.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.7 static void CMP_HAL_SetUnfilteredOutCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function is to switch to enable the filter for output of compare logic in CMP module. When enabled, it would set the unfiltered comparator output (CMPO) to equal COUT. When disabled, it would set the filtered comparator output(CMPO) to equal COUTA.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.8 static void CMP_HAL_SetInvertLogicCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function is to switch to enable the polarity of the analog comparator function. When enabled, it would invert the comparator output logic.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.9 static void CMP_HAL_SetHighSpeedCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function is to switch to enable the power (speed) comparison mode in CMP module. When enabled, it would select the High-Speed (HS) comparison mode. In this mode, CMP has faster output propagation delay and higher current consumption.

Parameters

CMP HAL Driver

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.10 static void CMP_HAL_SetSampleModeCmd (*uint32_t baseAddr, bool enable*) [*inline*], [*static*]

This function is to switch to enable the sample mode in CMP module. When any sampled mode is active, COUTA is sampled whenever a rising-edge of filter block clock input or WINDOW/SAMPLE signal is detected.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.11 static void CMP_HAL_SetFilterPeriodValue (*uint32_t baseAddr, uint8_t value*) [*inline*], [*static*]

This function is to set the filter sample period in CMP module. The setting value specifies the sampling period, in bus clock cycles, of the comparator output filter, when sample mode is disable. Setting the value to 0x0 would disable the filter. This API has no effect when sample mode is enabled. In that case, the external SAMPLE signal is used to determine the sampling period.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>value</i>	Count of bus clock cycles for per sample.

4.2.3.12 static bool CMP_HAL_GetOutputLogic (*uint32_t baseAddr*) [*inline*], [*static*]

This function is to get the comparator logic output in CMP module. It would return the current value of the analog comparator output. The value would be reset to 0 and read as de-assert value when the CMP module is disable. When setting to invert mode, the comparator logic output would be inverted as well.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The logic output is assert or not.

4.2.3.13 static bool CMP_HAL_GetOutputFallingFlag (uint32_t *baseAddr*) [inline], [static]

This function is to get the logic output's falling edge event in CMP module. It would detect a falling-edge on COUT and return the assert state when falling-edge on COUT has occurred.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The falling-edge on COUT has occurred or not.

4.2.3.14 static void CMP_HAL_ClearOutputFallingFlag (uint32_t *baseAddr*) [inline], [static]

This function is to clear the logic output's falling edge event in CMP module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

4.2.3.15 static bool CMP_HAL_GetOutputRisingFlag (uint32_t *baseAddr*) [inline], [static]

This function is to get the logic output's rising edge event in CMP module. It would detect a rising-edge on COUT and return the assert state when rising-edge on COUT has occurred.

Parameters

CMP HAL Driver

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The rising-edge on COUT has occurred or not.

4.2.3.16 static void CMP_HAL_ClearOutputRisingFlag (uint32_t *baseAddr*) [inline], [static]

This function is to clear the logic output's rising edge event in CMP module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

4.2.3.17 static void CMP_HAL_SetOutputFallingIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function is to switch to enable requesting interrupt when falling-edge on COUT has occurred. When enabled, it would generate a interrupt request when falling-edge on COUT has occurred.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.18 static bool CMP_HAL_GetOutputFallingIntCmd (uint32_t *baseAddr*) [inline], [static]

This function is to get the switcher of interrupt request on COUT's falling-edge. When it is assert, a interrupt request would be generated when falling-edge on COUT has occurred.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The status of switcher to enable the feature.

4.2.3.19 **static void CMP_HAL_SetOutputRisingIntCmd (uint32_t baseAddr, bool enable) [inline], [static]**

This function is to get the switcher of interrupt request on COUT's rising-edge. When it is assert, a interrupt request would be generated when rising-edge on COUT has occurred.

CMP HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.20 static bool CMP_HAL_GetOutputRisingIntCmd (uint32_t *baseAddr*) [inline], [static]

This function is to get the switcher of interrupt request on COUT's rising-edge. When it is assert, a interrupt request would be generated when rising-edge on COUT has occurred.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The status of switcher to enable the feature.

4.2.3.21 static void CMP_HAL_SetDacCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function is to switch to enable internal 6-bit DAC in CMP module. When enabled, the internal 6-bit DAC could be used as an input channel to the analog comparator.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature.

4.2.3.22 static void CMP_HAL_SetDacRefVoltSrcMode (uint32_t *baseAddr*, cmp_dac_ref_volt_src_mode_t *mode*) [inline], [static]

This function is to set the reference voltage source for internal 6-bit DAC in CMP module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of the feature, see to "cmp_dac_ref_volt_src_mode_t".

4.2.3.23 static void CMP_HAL_SetDacValue (*uint32_t baseAddr, uint8_t value*) [inline], [static]

This function is to set the output value for internal 6-bit DAC in CMP module. The output voltage of DAC would be $DACO = (V_{in}/64) * (value + 1)$, so the DACO range is from $V_{in}/64$ to V_{in} .

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>value</i>	Setting value, 6-bit available.

4.2.3.24 static void CMP_HAL_SetPlusInputChnMuxMode (*uint32_t baseAddr, cmp_chn_mux_mode_t mode*) [inline], [static]

This function is to set the plus channel for analog comparator. The input channels of plus and minus side are comes from the same channel mux. When an inappropriate operation selects the same input for both mux, the comparator automatically shuts down to prevent itself from becoming a noise generator. For channel's use cases, see to the speculation for each SOC.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Channel mux mode, see to "cmp_chn_mux_mode_t".

4.2.3.25 static void CMP_HAL_SetMinusInputChnMuxMode (*uint32_t baseAddr, cmp_chn_mux_mode_t mode*) [inline], [static]

This function is to set the minus channel for analog comparator. The input channels of plus and minus side are comes from the same channel mux. When an inappropriate operation selects the same input for both mux, the comparator automatically shuts down to prevent itself from becoming a noise generator. For channel's use cases, see to the speculation for each SOC.

Parameters

CMP HAL Driver

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Channel mux mode, see to "cmp_chn_mux_mode_t".

4.3 CMP Peripheral Driver

4.3.1 Overview

This section describes the programming interface of the CMP Peripheral driver. The CMP peripheral driver configures the CMP (Comparator). It handles initialization and configuration of CMP module.

4.3.2 CMP Driver model building

CMP driver has three parts:

- Basic Comparator - This part handles the mechanism that compares the voltage from the two input channels and outputs the assertion if the plus side voltage is higher than the minus side voltage.
- Internal 6-bit DAC - The internal 6-bit DAC can be configured as one of the input channel for the basic comparator. It can provide a reference voltage level when comparing with the other voltage signal.
- Sample/Filter - The Sample/Filter is an additional feature to improve the signal of the comparator's output. A sample clock, window mode and accumulation filter can be used to configure the Sample/Filter. When configuring the Sample/Filter, limited setting are available. See the "cmp_sample_filter_mode_t" definition in the "fsl_cmp_driver.h" file, or a chip reference manual for detailed information.

APIs are separate for each part and can be customized according to the application requirements.

4.3.3 CMP Call diagram

1. Ensure that the pin mux settings are ready before using the driver.
2. Call the "CMP_DRV_Init()" function to initialize the basic comparator. A configuration structure "cmp_user_config_t" type is required to keep the initialization information. The structure is populated by the application for a specific case, or by the "CMP_DRV_StructInitUserConfigDefault" API with available settings. A memory block is required and should be represented as a variable of the "cmp_state_t" type to keep state with the [CMP_DRV_Init\(\)](#) function.
3. Optionally, call the "CMP_DRV_EnableDac()" function to configure the internal 6-bit DAC if it is used as one of the input channels. A configuration structure of the "cmp_dac_config_t" type is required.
4. Optionally, call the "CMP_DRV_ConfigSampleFilter()" function to configure the Sample/Filter. A configuration structure of the "cmp_sample_filter_config_t" type is required keep the configuration.
5. Optionally, call the "CMP_DRV_InstallCallback" function to install the user-defined callback function into the interrupt routine service. When the CMP is configured to enable the interrupt, the installed callback function is called after the interrupt event occurs.
6. Finally, the CMP automatically responds to the external events.

This is an example to initialize and configure the CMP driver for typical use cases.

```
// cmp_test.c //
```

CMP Peripheral Driver

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "fsl_cmp_driver.h"
#include <board.h>
#include "fsl_device_registers.h"
#include "fsl_os_abstraction.h"

#define TEST_CMP_INSTANCE      BOARD_CMP_INSTANCE
#define TEST_TIME_OUT_MS       (4000U)

cmp_state_t TestCmpStateStruct;

// Normal Interrupt mode. //
volatile bool bRisingEvent = false;
volatile bool bFallingEvent = false;
volatile uint32_t TimeMs;

static void CMP_TEST_ISR(void);
static void CMP_TEST_InitIO(void);

extern void CMP_TEST_InstallCallback(uint32_t instance, void (*callbackFunc)(void) );
extern void configure_cmp_pins(uint32_t instance);

int main(void)
{
    cmp_user_config_t TestCmpUserConfigStruct;
    cmp_sample_filter_config_t TestCmpSampleFilterConfigStruct;
    cmp_dac_config_t TestCmpDacConfigStruct;

    hardware_init();
    dbg_uart_init();

    OSA_Init();

    printf("CMP PD TEST: Start...\r\n");

    // Init IO for CMP. //
    configure_cmp_pins(TEST_CMP_INSTANCE);

    // Init the CMP comparator. //
    CMP_DRV_StructInitUserConfigDefault(&TestCmpUserConfigStruct, (
        cmp_chn_mux_mode_t)BOARD_CMP_CHANNEL, kCmpInputChnDac);
    TestCmpUserConfigStruct.risingIntEnable = true;
    TestCmpUserConfigStruct.fallingIntEnable = true;
    CMP_DRV_Init(TEST_CMP_INSTANCE, &TestCmpUserConfigStruct, &TestCmpStateStruct);

    // Configure the internal DAC when in used. //
    TestCmpDacConfigStruct.dacValue = 32U; // 0U - 63U //
    TestCmpDacConfigStruct.refVoltSrcMode = kCmpDacRefVoltSrcOf2;
    CMP_DRV_EnableDac(TEST_CMP_INSTANCE, &TestCmpDacConfigStruct);

    // Configure the Sample/Filter Mode. //
    TestCmpSampleFilterConfigStruct.workMode = kCmpContinuousMode;
    CMP_DRV_ConfigSampleFilter(TEST_CMP_INSTANCE, &
        TestCmpSampleFilterConfigStruct);

    // Install the callback into interrupt. //
    CMP_TEST_InstallCallback(TEST_CMP_INSTANCE, CMP_TEST_ISR);

    // Start the CMP function. //
    CMP_DRV_Start(TEST_CMP_INSTANCE);

    printf("Please input signal in %d ms\r\n", TEST_TIME_OUT_MS);
    TimeMs = OSA_TimeGetMsec();

    while (1)
```

```

{
    if ( (TimeMs+TEST_TIME_OUT_MS) <= OSA_TimeGetMsec() )
    {
        break;
    }
    if (bRisingEvent)
    {
        printf("^\r CMP output rising event occur!\r\n");
        printf("  CMP output level %d\r\n", CMP_DRV_GetOutput(TEST_CMP_INSTANCE));
        bRisingEvent = false;
    }
    if (bFallingEvent)
    {
        printf("v\r CMP output failing event occur!\r\n");
        printf("  CMP output level %d\r\n", CMP_DRV_GetOutput(TEST_CMP_INSTANCE));
        bFallingEvent = false;
    }
}
}

CMP_DRV_Deinit(TEST_CMP_INSTANCE);
printf("CMP PD Test ");
printf("Succeed\r\n");
}

static void CMP_TEST_ISR(void)
{
    if (CMP_DRV_GetFlag(TEST_CMP_INSTANCE, kCmpFlagOfCoutRising) )
    {
        if (!bRisingEvent)
        {
            bRisingEvent = true;
        }
    }
    if (CMP_DRV_GetFlag(TEST_CMP_INSTANCE, kCmpFlagOfCoutFalling) )
    {
        if (!bFallingEvent)
        {
            bFallingEvent = true;
        }
    }
}
}

```

Data Structures

- struct [cmp_user_config_t](#)
Defines a structure for configuring the comparator in the CMP module. [More...](#)
- struct [cmp_sample_filter_config_t](#)
Defines a structure to configure the Window/Filter in CMP module. [More...](#)
- struct [cmp_dac_config_t](#)
Defines a structure to configure the internal DAC in CMP module. [More...](#)
- struct [cmp_state_t](#)
Internal driver state information. [More...](#)

CMP Peripheral Driver

Enumerations

- enum `cmp_sample_filter_mode_t` {
 `kCmpContinuousMode` = 0U,
 `kCmpSampleWithNoFilteredMode` = 1U,
 `kCmpSampleWithFilteredMode` = 2U,
 `kCmpWindowedMode` = 3U,
 `kCmpWindowedFilteredMode` = 4U }
 Definition selections of sample and filter mode in the CMP module.
- enum `cmp_flag_t` {
 `kCmpFlagOfCoutRising` = 0U,
 `kCmpFlagOfCoutFalling` = 1U }
 Defines type of flags for the CMP event.

Functions

- `cmp_status_t CMP_DRV_StructInitUserConfigDefault (cmp_user_config_t *userConfigPtr, cmp_chn_mux_mode_t plusInput, cmp_chn_mux_mode_t minusInput)`
 Populates the initial user configuration with default settings.
- `cmp_status_t CMP_DRV_Init (uint32_t instance, cmp_user_config_t *userConfigPtr, cmp_state_t *userStatePtr)`
 Initializes the CMP module.
- `void CMP_DRV_Deinit (uint32_t instance)`
 De-initializes the CMP module.
- `void CMP_DRV_Start (uint32_t instance)`
 Starts the CMP module.
- `void CMP_DRV_Stop (uint32_t instance)`
 Stops the CMP module.
- `cmp_status_t CMP_DRV_EnableDac (uint32_t instance, cmp_dac_config_t *dacConfigPtr)`
 Enables the internal DAC in the CMP module.
- `void CMP_DRV_DisableDac (uint32_t instance)`
 Disables the internal DAC in the CMP module.
- `cmp_status_t CMP_DRV_ConfigSampleFilter (uint32_t instance, cmp_sample_filter_config_t *configPtr)`
 Configures the Sample feature in the CMP module.
- `bool CMP_DRV_GetOutput (uint32_t instance)`
 Gets the output of the CMP module.
- `bool CMP_DRV_GetFlag (uint32_t instance, cmp_flag_t flag)`
 Gets the state of the CMP module.
- `void CMP_DRV_ClearFlag (uint32_t instance, cmp_flag_t flag)`
 Clears the event record of the CMP module.

Variables

- `const uint32_t g_cmpBaseAddr []`
 Table of base addresses for CMP instances.
- `const IRQn_Type g_cmplrqId [HW_CMP_INSTANCE_COUNT]`

Table to save CMP IRQ enumeration numbers defined in CMSIS header file.

4.3.4 Data Structure Documentation

4.3.4.1 struct cmp_user_config_t

This structure holds the configuration for the comparator inside the CMP module. With the configuration, the CMP can be set as a basic comparator without additional features.

Data Fields

- **cmp_hysteresis_mode_t hysteresisMode**
Set the hysteresis level.
- **bool pinoutEnable**
Enable outputting the CMPO to pin.
- **bool pinoutUnfilteredEnable**
Enable outputting unfiltered result to CMPO.
- **bool invertEnable**
Enable inverting the comparator's result.
- **bool highSpeedEnable**
Enable working in speed mode.
- **bool risingIntEnable**
Enable using CMPO rising interrupt.
- **bool fallingIntEnable**
Enable using CMPO falling interrupt.
- **cmp_chn_mux_mode_t plusChnMux**
Set the Plus side input to comparator.
- **cmp_chn_mux_mode_t minusChnMux**
Set the Minus side input to comparator.

CMP Peripheral Driver

4.3.4.1.0.4 Field Documentation

4.3.4.1.0.4.1 `cmp_hysteresis_mode_t cmp_user_config_t::hysteresisMode`

4.3.4.1.0.4.2 `bool cmp_user_config_t::pinoutEnable`

4.3.4.1.0.4.3 `bool cmp_user_config_t::pinoutUnfilteredEnable`

4.3.4.1.0.4.4 `bool cmp_user_config_t::invertEnable`

4.3.4.1.0.4.5 `bool cmp_user_config_t::highSpeedEnable`

4.3.4.1.0.4.6 `bool cmp_user_config_t::risingIntEnable`

4.3.4.1.0.4.7 `bool cmp_user_config_t::fallingIntEnable`

4.3.4.1.0.4.8 `cmp_chn_mux_mode_t cmp_user_config_t::plusChnMux`

4.3.4.1.0.4.9 `cmp_chn_mux_mode_t cmp_user_config_t::minusChnMux`

4.3.4.2 `struct cmp_sample_filter_config_t`

This structure holds the configuration for the Window/Filter inside the CMP module. With the configuration, the CMP module can operate in advanced mode.

Data Fields

- `cmp_sample_filter_mode_t workMode`
Sample/Filter's work mode.
- `bool useExtSampleOrWindow`
Switcher to use external WINDOW/SAMPLE signal.
- `uint8_t filterClkDiv`
Filter's prescaler which divides from the bus clock.
- `cmp_filter_counter_mode_t filterCount`
Sample count for filter.

4.3.4.2.0.5 Field Documentation

4.3.4.2.0.5.1 `cmp_sample_filter_mode_t cmp_sample_filter_config_t::workMode`

4.3.4.2.0.5.2 `bool cmp_sample_filter_config_t::useExtSampleOrWindow`

4.3.4.2.0.5.3 `uint8_t cmp_sample_filter_config_t::filterClkDiv`

4.3.4.2.0.5.4 `cmp_filter_counter_mode_t cmp_sample_filter_config_t::filterCount`

See "cmp_filter_counter_mode_t".

4.3.4.3 struct cmp_dac_config_t

This structure holds the configuration for DAC inside the CMP module. With the configuration, the internal DAC provides a reference voltage level and is chosen as the CMP input.

Data Fields

- `cmp_dac_ref_volt_src_mode_t refVoltSrcMode`
Select the reference voltage source for internal DAC.
- `uint8_t dacValue`
Set the value for internal DAC.

4.3.4.3.0.6 Field Documentation

4.3.4.3.0.6.1 `cmp_dac_ref_volt_src_mode_t cmp_dac_config_t::refVoltSrcMode`

4.3.4.3.0.6.2 `uint8_t cmp_dac_config_t::dacValue`

4.3.4.4 struct cmp_state_t

The contents of this structure are internal to the driver and should not be modified by users. Also, contents of the structure are subject to change in future releases.

4.3.5 Enumeration Type Documentation

4.3.5.1 enum cmp_sample_filter_mode_t

Comparator sample/filter is available in several modes. Use the enumeration to identify the comparator's status:

kCmpContinuousMode - Continuous Mode: Both window control and filter blocks are completely bypassed. The comparator output is updated continuously.
kCmpSampleWithNoFilteredMode - Sample, Non-Filtered Mode: Window control is completely bypassed. The comparator output is sampled whenever a rising-edge is detected on the filter block clock input. The filter clock prescaler can be configured as a divider from the bus clock.
kCmpSampleWithFilteredMode - Sample, Filtered Mode: Similar to "- Sample, Non-Filtered Mode", but the filter is active in this mode. The filter counter value also becomes configurable.
kCmpWindowedMode - Windowed Mode: In Windowed Mode, only output of analog comparator is passed when the WINDOW signal is high. The last latched value is held when the WINDOW signal is low.
kCmpWindowedFilteredMode - Window/Filtered Mode: This is a complex mode because it uses both window and filtering features. It also has the highest latency of all modes. This can be approximated to up to 1 bus clock synchronization in the window function

- ((filter counter * filter prescaler) + 1) bus clock for the filter function.

Enumerator

kCmpContinuousMode Continuous Mode.

CMP Peripheral Driver

kCmpSampleWithNoFilteredMode Sample, Non-Filtered Mode.

kCmpSampleWithFilteredMode Sample, Filtered Mode.

kCmpWindowedMode Window Mode.

kCmpWindowedFilteredMode Window/Filtered Mode.

4.3.5.2 enum cmp_flag_t

Enumerator

kCmpFlagOfCoutRising Identifier to indicate if the COUT change from logic zero to one.

kCmpFlagOfCoutFalling Identifier to indicate if the COUT change from logic one to zero.

4.3.6 Function Documentation

4.3.6.1 cmp_status_t CMP_DRV_StructInitUserConfigDefault (cmp_user_config_t * *userConfigPtr*, cmp_chn_mux_mode_t *plusInput*, cmp_chn_mux_mode_t *minusInput*)

This function populates the initial user configuration with default settings. The default settings enable the CMP module to operate as a comparator. The settings are :

- .hystersisMode = kCmpHystersisOfLevel0
- .pinoutEnable = true
- .pinoutUnfilteredEnable = true
- .invertEnable = false
- .highSpeedEnable = false
- .dmaEnable = false
- .risingIntEnable = false
- .fallingIntEnable = false
- .triggerEnable = false However, it is still recommended to fill some fields of structure such as channel mux according to the application requirements. Note that this function does not set the configuration to hardware.

Parameters

<i>userConfigPtr</i>	Pointer to structure of configuration. See "cmp_user_config_t".
<i>plusInput</i>	Plus Input mux selection. See "cmp_chn_mux_mode_t".
<i>minusInput</i>	Minus Input mux selection. See "cmp_chn_mux_mode_t".

Returns

Execution status.

4.3.6.2 `cmp_status_t CMP_DRV_Init (uint32_t instance, cmp_user_config_t * userConfigPtr, cmp_state_t * userStatePtr)`

This function initializes the CMP module, enables the clock and sets the interrupt switcher. The CMP module is configured as a basic comparator.

CMP Peripheral Driver

Parameters

<i>instance</i>	CMP instance ID.
<i>userConfigPtr</i>	Pointer to structure of configuration. See "cmp_user_config_t".
<i>userStatePtr</i>	Pointer to structure of context. See "cmp_state_t".

Returns

Execution status.

4.3.6.3 void CMP_DRV_Deinit (uint32_t *instance*)

This function de-initializes the CMP module. It shuts down the CMP clock and disables the interrupt. This API should be called when CMP is no longer used in the application. It also reduces power consumption.

Parameters

<i>instance</i>	CMP instance ID.
-----------------	------------------

4.3.6.4 void CMP_DRV_Start (uint32_t *instance*)

This function starts the CMP module. The configuration does not take effect until the module is started.

Parameters

<i>instance</i>	CMP instance ID.
-----------------	------------------

4.3.6.5 void CMP_DRV_Stop (uint32_t *instance*)

This function stops the CMP module. Note that this function does not shut down the module, but only pauses the features.

Parameters

<i>instance</i>	CMP instance ID.
-----------------	------------------

4.3.6.6 cmp_status_t CMP_DRV_EnableDac (uint32_t *instance*, cmp_dac_config_t * *dacConfigPtr*)

This function enables the internal DAC in the CMP module. It takes effect only when the internal DAC has been chosen as an input channel for the comparator. Then, the DAC channel can be programmed to

provide a reference voltage level.

CMP Peripheral Driver

Parameters

<i>instance</i>	CMP instance ID.
<i>dacConfigPtr</i>	Pointer to structure of configuration. See "cmp_dac_config_t".

Returns

Execution status.

4.3.6.7 void CMP_DRV_DisableDac (uint32_t *instance*)

This function disables the internal DAC in the CMP module. It should be called if the internal DAC is no longer used by the application.

Parameters

<i>instance</i>	CMP instance ID.
-----------------	------------------

4.3.6.8 cmp_status_t CMP_DRV_ConfigSampleFilter (uint32_t *instance*, cmp_sample_filter_config_t * *configPtr*)

This function configures the CMP working in Sample modes. These modes are advanced features in addition to the basic comparator such as Window Mode, Filter Mode, etc. See "cmp_sample_filter_config_t" for detailed description.

Parameters

<i>instance</i>	CMP instance ID.
<i>configPtr</i>	Pointer to a structure of configurations. See "cmp_sample_filter_config_t".

Returns

Execution status.

4.3.6.9 bool CMP_DRV_GetOutput (uint32_t *instance*)

This function gets the output of the CMP module. The output source depends on the configuration when initializing the comparator. When the [cmp_user_config_t.pinoutUnfilteredEnable](#) is false, the output is processed by the filter. Otherwise, the output is that the signal did not pass the filter.

Parameters

<i>instance</i>	CMP instance ID.
-----------------	------------------

Returns

Output logic's assertion. When not inverted, plus side > minus side, it is true.

4.3.6.10 **bool CMP_DRV_GetFlag (uint32_t *instance*, cmp_flag_t *flag*)**

This function gets the state of the CMP module. It returns if the indicated event has been detected.

Parameters

<i>instance</i>	CMP instance ID.
<i>flag</i>	Represent events or states. See "cmp_flag_t".

Returns

Assertion if indicated event occurs.

4.3.6.11 **void CMP_DRV_ClearFlag (uint32_t *instance*, cmp_flag_t *flag*)**

This function clears the event record of the CMP module.

Parameters

<i>instance</i>	CMP instance ID.
<i>flag</i>	Represent events or states. See "cmp_flag_t".

4.3.7 Variable Documentation

4.3.7.1 **const uint32_t g_cmpBaseAddr[]**

4.3.7.2 **const IRQn_Type g_cmplrqld[HW_CMP_INSTANCE_COUNT]**

Chapter 5

Watchdog Timer

5.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the COP Watchdog Timer (WDOG) block of Kinetis devices.

Modules

- COP HAL driver

COP HAL driver

5.2 COP HAL driver

5.2.1 Overview

The section describes the programming interface of the COP HAL driver.

Data Structures

- union `cop_common_config_t`
Define COP common configure. [More...](#)

Enumerations

- enum `cop_clock_source_t` {
 `kCopLpoClock`,
 `kCopBusClock` }
COP clock source selection.
- enum `cop_timeout_cycles_t` {
 `kCopTimeout_short_2to5_or_long_2to13` = 1U,
 `kCopTimeout_short_2to8_or_long_2to16` = 2U,
 `kCopTimeout_short_2to10_or_long_2to18` = 3U }
Define the value of the COP timeout cycles.

COP HAL.

- static void `COP_HAL_SetCommonConfiguration` (uint32_t baseAddr, `cop_common_config_t` copConfiguration)
config the COP watchdog.
- static void `COP_HAL_Disable` (uint32_t baseAddr)
disable the COP watchdog.
- static bool `COP_HAL_IsEnabled` (uint32_t baseAddr)
Determines whether the COP is enabled.
- static `cop_clock_source_t` `COP_HAL_GetClockSource` (uint32_t baseAddr)
Gets the COP clock source.
- static `cop_timeout_cycles_t` `COP_HAL_GetTimeoutCycles` (uint32_t baseAddr)
Returns timeout of COP watchdog .
- static bool `COP_HAL_GetWindowModeCmd` (uint32_t baseAddr)
get the COP watchdog run mode—window mode or normal window.
- static void `COP_HAL_Refresh` (uint32_t baseAddr)
Servicing COP watchdog.
- static void `COP_HAL_ResetSystem` (uint32_t baseAddr)
Reset system.
- void `COP_HAL_Init` (uint32_t baseAddr)
Restores the COP module to reset value.

5.2.2 Data Structure Documentation

5.2.2.1 union cop_common_config_t

5.2.3 Enumeration Type Documentation

5.2.3.1 enum cop_clock_source_t

Enumerator

kCopLpoClock LPO clock,1K HZ.

kCopBusClock BUS clock.

5.2.3.2 enum cop_timeout_cycles_t

Enumerator

kCopTimeout_short_2to5_or_long_2to13 2 to 5 clock cycles when clock source is LPO or in short timeout mode otherwise 2 to 13 clock cycles

kCopTimeout_short_2to8_or_long_2to16 2 to 8 clock cycles when clock source is LPO or in short timeout mode otherwise 2 to 16 clock cycles

kCopTimeout_short_2to10_or_long_2to18 2 to 10 clock cycles when clock source is LPO or in short timeout mode otherwise 2 to 18 clock cycles

5.2.4 Function Documentation

5.2.4.1 static void COP_HAL_SetCommonConfiguration (uint32_t baseAddr, cop_common_config_t copConfiguration) [inline], [static]

The COP Control register is write once after reset.

Parameters

<i>baseAddr</i>	The COP peripheral base address
<i>cop-Configuration</i>	configure COP control register

5.2.4.2 static void COP_HAL_Disable(uint32_t baseAddr) [inline], [static]

This function is used to disable the COP watchdog and should be called after reset if your application does not need COP watchdog.

COP HAL driver

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

5.2.4.3 static bool COP_HAL_IsEnabled (uint32_t *baseAddr*) [inline], [static]

This function is used to check whether the COP is running.

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

Return values

<i>true</i>	COP is enabled
<i>false</i>	COP is disabled

5.2.4.4 static cop_clock_source_t COP_HAL_GetClockSource (uint32_t *baseAddr*) [inline], [static]

This function is used to get the COP clock source.

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

Return values

<i>clockSource</i>	COP clock source, see cop_clock_source_t
--------------------	--

5.2.4.5 static cop_timeout_cycles_t COP_HAL_GetTimeoutCycles (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

Returns

COP timeout cycles

5.2.4.6 **static bool COP_HAL_GetWindowModeCmd (uint32_t *baseAddr*) [inline],
[static]**

COP HAL driver

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

Return values

0	COP watchdog run in normal mode
1	COP watchdog run in window mode

5.2.4.7 static void COP_HAL_Refresh (uint32_t *baseAddr*) [inline], [static]

This function reset COP timeout by write 0x55 then 0xAA, write any other value will generate a system reset. The writing operations should be atomic.

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

5.2.4.8 static void COP_HAL_ResetSystem (uint32_t *baseAddr*) [inline], [static]

This function reset system

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

5.2.4.9 void COP_HAL_Init (uint32_t *baseAddr*)

This function restores the COP module to reset value.

Parameters

<i>baseAddr</i>	The COP peripheral base address
-----------------	---------------------------------

Chapter 6

12-bit Cyclic Analog-to-Digital Converter (CADC)

6.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the 12-bit Cyclic Analog-to-Digital Converter (CADC) block of Kinetis devices.

Modules

- [CADC HAL Driver](#)
- [CADC Peripheral Driver](#)

6.2 CADC HAL Driver

6.2.1 Overview

This section describes the programming interface of the CADC HAL driver.

Enumerations

- enum `cadc_status_t` {
 `kStatus_CADC_Success` = 0U,
 `kStatus_CADC_InvalidArgument` = 1U,
 `kStatus_CADC_Failed` = 2U }
 CADC status return codes.
- enum `cadc_conv_id_t` {
 `kCAdcConvA` = 0U,
 `kCAdcConvB` = 1U }
 Defines the type of enumerating ADC converter's ID.
- enum `cadc_diff_chn_mode_t` {
 `kCAdcDiffChnANA0_1` = 0U,
 `kCAdcDiffChnANA2_3` = 1U,
 `kCAdcDiffChnANA4_5` = 2U,
 `kCAdcDiffChnANA6_7` = 3U,
 `kCAdcDiffChnANB0_1` = 4U,
 `kCAdcDiffChnANB2_3` = 5U,
 `kCAdcDiffChnANB4_5` = 6U,
 `kCAdcDiffChnANB6_7` = 7U }
 Defines the type of enumerating ADC differential channel pair.
- enum `cadc_chn_sel_mode_t` {
 `kCAdcChnSelN` = 0U,
 `kCAdcChnSelP` = 1U,
 `kCAdcChnSelBoth` = 2U }
 Defines the type of enumerating ADC channel in differential pair.
- enum `cadc_scan_mode_t` {
 `kCAdcScanOnceSequential` = 0U,
 `kCAdcScanOnceParallel` = 1U,
 `kCAdcScanLoopSequential` = 2U,
 `kCAdcScanLoopParallel` = 3U,
 `kCAdcScanTriggeredSequential` = 4U,
 `kCAdcScanTriggeredParalled` = 5U }
 Defines the type of enumerating ADC converter's scan mode.
- enum `cadc_zero_crossing_mode_t` {
 `kCAdcZeroCrossingDisable` = 0U,
 `kCAdcZeroCrossingAtRisingEdge` = 1U,
 `kCAdcZeroCrossingAtFallingEdge` = 2U,
 `kCAdcZeroCrossingAtBothEdge` = 3U }
 Defines the type of enumerating zero crossing detection mode for each slot.

- enum `cadc_gain_mode_t` {

 kCAdcSGainBy1 = 0U,

 kCAdcSGainBy2 = 1U,

 kCAdcSGainBy4 = 2U }

 Defines the type of enumerating amplification mode for each slot.
- enum `cadc_conv_speed_mode_t` {

 kCAdcConvClkLimitBy6_25MHz = 0U,

 kCAdcConvClkLimitBy12_5MHz = 1U,

 kCAdcConvClkLimitBy18_75MHz = 2U,

 kCAdcConvClkLimitBy25MHz = 3U }

 Defines the type of enumerating speed mode for each converter.
- enum `cadc_dma_trigger_src_mode_t` {

 kCAdcDmaTriggeredByEndOfScan = 0U,

 kCAdcDmaTriggeredByConvReady = 1U }

 Defines the type of DMA trigger source mode for each converter.

Functions

- void `CADC_HAL_Init` (uint32_t baseAddr)

 Initialize all the ADC registers to a known state.
- void `CADC_HAL_SetConvDmaCmd` (uint32_t baseAddr, `cadc_conv_id_t` convId, bool enable)

 Switch to enable DMA for ADC converter.
- void `CADC_HAL_SetConvStopCmd` (uint32_t baseAddr, `cadc_conv_id_t` convId, bool enable)

 Switch to enable DMA for ADC converter.
- void `CADC_HAL_SetConvStartCmd` (uint32_t baseAddr, `cadc_conv_id_t` convId)

 Trigger the ADC converter's conversion by software.
- void `CADC_HAL_SetConvSyncCmd` (uint32_t baseAddr, `cadc_conv_id_t` convId, bool enable)

 Switch to enable input SYNC signal to trigger conversion.
- void `CADC_HAL_SetConvEndOfScanIntCmd` (uint32_t baseAddr, `cadc_conv_id_t` convId, bool enable)

 Switch to enable interrupt caused by end of scan for each converter.
- static void `CADC_HAL_SetZeroCrossingIntCmd` (uint32_t baseAddr, bool enable)

 Switch to enable interrupt caused by zero crossing detection for ADC module.
- static void `CADC_HAL_SetLowLimitIntCmd` (uint32_t baseAddr, bool enable)

 Switch to enable interrupt caused by meet low limit for ADC module.
- static void `CADC_HAL_SetHighLimitIntCmd` (uint32_t baseAddr, bool enable)

 Switch to enable interrupt caused by meet high limit for ADC module.
- void `CADC_HAL_SetChnDiffCmd` (uint32_t baseAddr, `cadc_diff_chn_mode_t` chns, bool enable)

 Switch to enable differential sample mode for ADC conversion channel.
- static void `CADC_HAL_SetScanMode` (uint32_t baseAddr, `cadc_scan_mode_t` mode)

 Configure the scan mode for ADC module.
- static void `CADC_HAL_SetParallelSimultCmd` (uint32_t baseAddr, bool enable)

 Switch to enable simultaneous mode for ADC module.
- void `CADC_HAL_SetConvClkDiv` (uint32_t baseAddr, `cadc_conv_id_t` convId, uint16_t divider)

 Set divider for conversion clock from system clock.
- void `CADC_HAL_SetSlotZeroCrossingMode` (uint32_t baseAddr, uint32_t slotNum, `cadc_zero_crossing_mode_t` mode)

 Set zero crossing detection for each slot in conversion sequence.

CADC HAL Driver

- void **CADC_HAL_SetSlotSampleChn** (uint32_t baseAddr, uint32_t slotNum, **cadc_diff_chn_mode_t** diffChns, **cadc_chn_sel_mode_t** selMode)
Set sample channel for each slot in conversion sequence.
- void **CADC_HAL_SetSlotSampleEnableCmd** (uint32_t baseAddr, uint32_t slotNum, bool enable)
Switch to enable sample for each slot in conversion sequence.
- bool **CADC_HAL_GetConvInProgressFlag** (uint32_t baseAddr, **cadc_conv_id_t** convId)
Check if the conversion is in process for each converter.
- bool **CADC_HAL_GetConvEndOfScanIntFlag** (uint32_t baseAddr, **cadc_conv_id_t** convId)
Check if the end of scan event is asserted.
- void **CADC_HAL_ClearConvEndOfScanIntFlag** (uint32_t baseAddr, **cadc_conv_id_t** convId)
Clear the end of scan flag.
- static bool **CADC_HAL_GetZeroCrossingIntFlag** (uint32_t baseAddr)
Check if main zero crossing event is asserted.
- static void **CADC_HAL_ClearAllZeroCrossingFlag** (uint32_t baseAddr)
Clear all zero crossing flag.
- static bool **CADC_HAL_GetSlotZeroCrossingFlag** (uint32_t baseAddr, uint32_t slotNum)
Check if the slot's crossing event flag is asserted.
- static void **CADC_HAL_ClearSlotZeroCrossingFlag** (uint32_t baseAddr, uint32_t slotNum)
Clear the crossing event flag for slot.
- static bool **CADC_HAL_GetLowLimitIntFlag** (uint32_t baseAddr)
Check if main low limit event is asserted.
- static void **CADC_HAL_ClearAllLowLimitFlag** (uint32_t baseAddr)
Clear all the low limit flag.
- static bool **CADC_HAL_GetSlotLowLimitFlag** (uint32_t baseAddr, uint32_t slotNum)
Check if slot's low limit flag is asserted.
- static void **CADC_HAL_ClearSlotLowLimitFlag** (uint32_t baseAddr, uint32_t slotNum)
Clear slot's low limit flag.
- static bool **CADC_HAL_GetHighLimitIntFlag** (uint32_t baseAddr)
Check if the main high limit event is asserted.
- static void **CADC_HAL_ClearAllHighLimitFlag** (uint32_t baseAddr)
Clear all the high limit flag.
- static bool **CADC_HAL_GetSlotHighLimitFlag** (uint32_t baseAddr, uint32_t slotNum)
Check if slot's high limit flag is asserted.
- static void **CADC_HAL_ClearSlotHighLimitFlag** (uint32_t baseAddr, uint32_t slotNum)
Clear slot's high limit flag.
- static bool **CADC_HAL_GetSlotValueNegativeSign** (uint32_t baseAddr, uint32_t slotNum)
Check if the slot's conversion value is negative.
- static uint16_t **CADC_HAL_GetSlotValueData** (uint32_t baseAddr, uint32_t slotNum)
Get the slot's conversion value without sign.
- static void **CADC_HAL_SetSlotLowLimitValue** (uint32_t baseAddr, uint32_t slotNum, uint16_t val)
Set the slot's conversion low limit value.
- static uint16_t **CADC_HAL_GetSlotLowLimitValue** (uint32_t baseAddr, uint32_t slotNum)
Get the slot's conversion low limit value.
- static void **CADC_HAL_SetSlotHighLimitValue** (uint32_t baseAddr, uint32_t slotNum, uint16_t val)
Set the slot's conversion high limit value.
- static uint16_t **CADC_HAL_GetSlotHighLimitValue** (uint32_t baseAddr, uint32_t slotNum)
Get the slot's conversion high limit value.
- static void **CADC_HAL_SetSlotOffsetValue** (uint32_t baseAddr, uint32_t slotNum, uint16_t val)
Set the slot's conversion high limit value.

- static uint16_t **CADC_HAL_GetSlotOffsetValue** (uint32_t baseAddr, uint32_t slotNum)
Get the slot's conversion high limit value.
- static void **CADC_HAL_SetAutoStandbyCmd** (uint32_t baseAddr, bool enable)
Switch to enable auto-standby mode for ADC module.
- bool **CADC_HAL_GetConvPowerUpFlag** (uint32_t baseAddr, **cadc_conv_id_t** convId)
Check if the ADC converter is powered up.
- static void **CADC_HAL_SetPowerUpDelayClk** (uint32_t baseAddr, uint16_t val)
Set the count power up delay of ADC clock for all the converter.
- static uint16_t **CADC_HAL_GetPowerUpDelayClk** (uint32_t baseAddr)
Get the count power up delay of ADC clock for all the converter.
- static void **CADC_HAL_SetAutoPowerDownCmd** (uint32_t baseAddr, bool enable)
Switch to enable auto-powerdown mode for ADC module.
- void **CADC_HAL_SetConvPowerUpCmd** (uint32_t baseAddr, **cadc_conv_id_t** convId, bool enable)
Switch to enable power up for each ADC converter.
- void **CADC_HAL_SetConvUseChnVrefHCmd** (uint32_t baseAddr, **cadc_conv_id_t** convId, bool enable)
Switch to use channel 3 as VrefH for each ADC converter.
- void **CADC_HAL_SetConvUseChnVrefLCmd** (uint32_t baseAddr, **cadc_conv_id_t** convId, bool enable)
Switch to use channel 2 as VrefL for each ADC converter.
- void **CADC_HAL_SetChnGainMode** (uint32_t baseAddr, uint32_t chnNum, **cadc_gain_mode_t** mode)
Set the amplification for each input channel.
- void **CADC_HAL_SetSlotSyncPointCmd** (uint32_t baseAddr, uint32_t slotNum, bool enable)
Switch to enable the sync point for each slot in sequence.
- void **CADC_HAL_SetConvSpeedLimitMode** (uint32_t baseAddr, **cadc_conv_id_t** convId, **cadc_conv_speed_mode_t** mode)
Set the conversion speed control mode for each ADC converter.
- static void **CADC_HAL_SetDmaTriggerSrcMode** (uint32_t baseAddr, **cadc_dma_trigger_src_mode_t** mode)
Set the DMA trigger mode for ADC module.
- void **CADC_HAL_SetConvSampleWindow** (uint32_t baseAddr, **cadc_conv_id_t** convId, uint16_t val)
Set the conversion speed control mode for each ADC converter.
- void **CADC_HAL_SetSlotScanIntCmd** (uint32_t baseAddr, uint32_t slotNum, bool enable)
Switch to enable scan interrupt for each slot in sequence.

6.2.2 Enumeration Type Documentation

6.2.2.1 enum **cadc_status_t**

Enumerator

- kStatus_CADC_Success** Success.
kStatus_CADC_InvalidArgument Invalid argument existed.
kStatus_CADC_Failed Execution failed.

CADC HAL Driver

6.2.2.2 enum `cadc_conv_id_t`

Enumerator

- kCAdcConvA* ID for ADC converter A.
- kCAdcConvB* ID for ADC converter B.

6.2.2.3 enum `cadc_diff_chn_mode_t`

Note, "cadc_diff_chn_mode_t" and "cadc_chn_sel_mode_t" can determine to select the single ADC sample channel.

Enumerator

- kCAdcDiffChnANA0_1* ConvA's Chn 0 & 1.
- kCAdcDiffChnANA2_3* ConvA's Chn 2 & 3.
- kCAdcDiffChnANA4_5* ConvA's Chn 4 & 5.
- kCAdcDiffChnANA6_7* ConvA's Chn 6 & 7.
- kCAdcDiffChnANB0_1* ConvB's Chn 0 & 1.
- kCAdcDiffChnANB2_3* ConvB's Chn 2 & 3.
- kCAdcDiffChnANB4_5* ConvB's Chn 4 & 5.
- kCAdcDiffChnANB6_7* ConvB's Chn 6 & 7.

6.2.2.4 enum `cadc_chn_sel_mode_t`

Note, "cadc_diff_chn_mode_t" and "cadc_chn_sel_mode_t" can determine to select the single ADC sample channel.

Enumerator

- kCAdcChnSelN* Select negative side channel.
- kCAdcChnSelP* Select positive side channel.
- kCAdcChnSelBoth* Select both of them in differential mode.

6.2.2.5 enum `cadc_scan_mode_t`

See to ADC_CTRL1[SMODE] from Reference Manual for more detailed information about ADC converter's scan mode.

Enumerator

- kCAdcScanOnceSequential* Once (single) sequential.
- kCAdcScanOnceParallel* Once parallel.
- kCAdcScanLoopSequential* Loop sequential.

kCAdcScanLoopParallel Loop parallel.

kCAdcScanTriggeredSequential Triggered sequential.

kCAdcScanTriggeredParalled Triggered parallel (default).

6.2.2.6 enum `cadc_zero_crossing_mode_t`

Enumerator

kCAdcZeroCrossingDisable Zero crossing detection disabled.

kCAdcZeroCrossingAtRisingEdge Enable for positive to negative sign change.

kCAdcZeroCrossingAtFallingEdge Enable for negative to positive sign change.

kCAdcZeroCrossingAtBothEdge Enable for any sign change.

6.2.2.7 enum `cadc_gain_mode_t`

Enumerator

kCAdcSGainBy1 x1 amplification.

kCAdcSGainBy2 x2 amplification.

kCAdcSGainBy4 x4 amplification.

6.2.2.8 enum `cadc_conv_speed_mode_t`

These items represent the clock speed at which the ADC converter can operate. Faster conversion speeds require greater current consumption.

Enumerator

kCAdcConvClkLimitBy6_25MHz Conversion clock frequency <= 6.25 MHz; current consumption per converter = 6 mA.

kCAdcConvClkLimitBy12_5MHz Conversion clock frequency <= 12.5 MHz; current consumption per converter = 10.8 mA.

kCAdcConvClkLimitBy18_75MHz Conversion clock frequency <= 18.75 MHz; current consumption per converter = 18 mA.

kCAdcConvClkLimitBy25MHz Conversion clock frequency <= 25 MHz; current consumption per converter = 25.2 mA.

6.2.2.9 enum `cadc_dma_trigger_src_mode_t`

During sequential and simultaneous parallel scan modes, it selects between end of scan for ConvA's scan and RDY status as the DMA source. During non-simultaneous parallel scan mode it selects between end of scan for converters A and B, and the RDY status as the DMA source

CADC HAL Driver

Enumerator

- kCAdcDmaTriggeredByEndOfScan* DMA trigger source is end of scan interrupt.
- kCAdcDmaTriggeredByConvReady* DMA trigger source is RDY status.

6.2.3 Function Documentation

6.2.3.1 void CADC_HAL_Init (*uint32_t baseAddr*)

The initial states of ADC registers are set as the Reference Manual describes.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

6.2.3.2 void CADC_HAL_SetConvDmaCmd (*uint32_t baseAddr, cadc_conv_id_t convId, bool enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".
<i>enable</i>	Assertion to enable the feature.

6.2.3.3 void CADC_HAL_SetConvStopCmd (*uint32_t baseAddr, cadc_conv_id_t convId, bool enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".
<i>enable</i>	Assertion to enable the feature.

6.2.3.4 void CADC_HAL_SetConvStartCmd (*uint32_t baseAddr, cadc_conv_id_t convId*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".

6.2.3.5 void CADC_HAL_SetConvSyncCmd (uint32_t *baseAddr*, **cadc_conv_id_t convId**, bool *enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".
<i>enable</i>	Assertion to enable the feature.

6.2.3.6 void CADC_HAL_SetConvEndOfScanIntCmd (uint32_t *baseAddr*, **cadc_conv_id_t convId**, bool *enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".
<i>enable</i>	Assertion to enable the feature.

6.2.3.7 static void CADC_HAL_SetZeroCrossingIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Assertion to enable the feature.

6.2.3.8 static void CADC_HAL_SetLowLimitIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Assertion to enable the feature.

6.2.3.9 static void CADC_HAL_SetHighLimitIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Assertion to enable the feature.

6.2.3.10 void CADC_HAL_SetChnDiffCmd (uint32_t *baseAddr*, cadc_diff_chn_mode_t *chns*, bool *enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chns</i>	Selection of differential channel pair, see to "cadc_diff_chn_mode_t".
<i>enable</i>	Assertion to enable the feature.

6.2.3.11 static void CADC_HAL_SetScanMode (uint32_t *baseAddr*, cadc_scan_mode_t *mode*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of scan mode, see to "cadc_scan_mode_t".

6.2.3.12 static void CADC_HAL_SetParallelSimultCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Assertion to enable the feature.

6.2.3.13 void CADC_HAL_SetConvClkDiv (uint32_t *baseAddr*, cadc_conv_id_t *convId*, uint16_t *divider*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".
<i>divider</i>	6-bit divider value.

6.2.3.14 void CADC_HAL_SetSlotZeroCrossingMode (uint32_t *baseAddr*, uint32_t *slotNum*, cadc_zero_crossing_mode_t *mode*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>mode</i>	Zero crossing detection mode, see to "cadc_zero_crossing_mode_t".

6.2.3.15 void CADC_HAL_SetSlotSampleChn (uint32_t *baseAddr*, uint32_t *slotNum*, cadc_diff_chn_mode_t *diffChns*, cadc_chn_sel_mode_t *selMode*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>diffChns</i>	Selection of differential channel pair, see to "cadc_diff_chn_mode_t".
<i>selMode</i>	Selection of each channel in differential channel pair, see to "cadc_chn_sel_mode_t".

6.2.3.16 void CADC_HAL_SetSlotSampleEnableCmd (uint32_t *baseAddr*, uint32_t *slotNum*, bool *enable*)

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>enable</i>	Assertion to enable the feature.

6.2.3.17 **bool CADC_HAL_GetConvInProgressFlag (uint32_t *baseAddr*, *cadc_conv_id_t convId*)**

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".

Returns

Assertion of indicated event.

6.2.3.18 **bool CADC_HAL_GetConvEndOfScanIntFlag (uint32_t *baseAddr*, *cadc_conv_id_t convId*)**

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".

Returns

Assertion of indicated event.

6.2.3.19 **void CADC_HAL_ClearConvEndOfScanIntFlag (uint32_t *baseAddr*, *cadc_conv_id_t convId*)**

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".

**6.2.3.20 static bool CADC_HAL_GetZeroCrossingIntFlag (uint32_t *baseAddr*)
[inline], [static]**

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

assertion of indicated event.

**6.2.3.21 static void CADC_HAL_ClearAllZeroCrossingFlag (uint32_t *baseAddr*)
[inline], [static]**

This operation will clear the main zero crossing event's flag.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

**6.2.3.22 static bool CADC_HAL_GetSlotZeroCrossingFlag (uint32_t *baseAddr*, uint32_t
slotNum) [inline], [static]**

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

assertion of indicated event.

**6.2.3.23 static void CADC_HAL_ClearSlotZeroCrossingFlag (uint32_t *baseAddr*, uint32_t
slotNum) [inline], [static]**

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

6.2.3.24 static bool CADC_HAL_GetLowLimitIntFlag (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

Assertion of indicated event.

6.2.3.25 static void CADC_HAL_ClearAllLowLimitFlag (uint32_t *baseAddr*) [inline], [static]

This operation will clear the main low limit event's flag.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

6.2.3.26 static bool CADC_HAL_GetSlotLowLimitFlag (uint32_t *baseAddr*, uint32_t *slotNum*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

Assertion of indicated event.

6.2.3.27 static void CADC_HAL_ClearSlotLowLimitFlag (uint32_t *baseAddr*, uint32_t *slotNum*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

6.2.3.28 static bool CADC_HAL_GetHighLimitIntFlag (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

Assertion of indicated event.

6.2.3.29 static void CADC_HAL_ClearAllHighLimitFlag (uint32_t *baseAddr*) [inline], [static]

This operation will clear the main high limit event's flag.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

6.2.3.30 static bool CADC_HAL_GetSlotHighLimitFlag (uint32_t *baseAddr*, uint32_t *slotNum*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

Assertion of indicated event.

6.2.3.31 static void CADC_HAL_ClearSlotHighLimitFlag (uint32_t *baseAddr*, uint32_t *slotNum*) [inline], [static]

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

6.2.3.32 static bool CADC_HAL_GetSlotValueNegativeSign (*uint32_t baseAddr, uint32_t slotNum*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

Assertion of indicated event.

6.2.3.33 static uint16_t CADC_HAL_GetSlotValueData (*uint32_t baseAddr, uint32_t slotNum*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

12-bit slot's conversion value.

6.2.3.34 static void CADC_HAL_SetSlotLowLimitValue (*uint32_t baseAddr, uint32_t slotNum, uint16_t val*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>val</i>	12-bit slot's conversion low limit value.

6.2.3.35 static uint16_t CADC_HAL_GetSlotLowLimitValue (uint32_t *baseAddr*, uint32_t *slotNum*) [inline], [static]

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

12-bit slot's conversion low limit value.

6.2.3.36 static void CADC_HAL_SetSlotHighLimitValue (*uint32_t baseAddr, uint32_t slotNum, uint16_t val*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>val</i>	12-bit slot's conversion high limit value.

6.2.3.37 static uint16_t CADC_HAL_GetSlotHighLimitValue (*uint32_t baseAddr, uint32_t slotNum*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

12-bit slot's conversion high limit value.

6.2.3.38 static void CADC_HAL_SetSlotOffsetValue (*uint32_t baseAddr, uint32_t slotNum, uint16_t val*) [inline], [static]

The value of the offset is used to correct the ADC result before it is stored in the result registers. The offset value is subtracted from the ADC result.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>val</i>	12-bit slot's conversion high limit value.

6.2.3.39 static uint16_t CADC_HAL_GetSlotOffsetValue (uint32_t *baseAddr*, uint32_t *slotNum*) [inline], [static]

The value of the offset is used to correct the ADC result before it is stored in the result registers. The offset value is subtracted from the ADC result.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.

Returns

12-bit slot's conversion high limit value.

6.2.3.40 static void CADC_HAL_SetAutoStandbyCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This API selects auto-standby mode. Auto-standby's configuration is ignored if auto-powerdown mode is active. When the ADC is idle, auto-standby mode selects the standby clock as the ADC clock source and puts the converters into standby current mode. At the start of any scan, the conversion clock is selected as the ADC clock and then a delay of some ADC clock cycles is imposed for current levels to stabilize. After this delay, the ADC will initiate the scan. When the ADC returns to the idle state, the standby clock is again selected and the converters revert to the standby current state.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Assertion to enable the feature.

6.2.3.41 bool CADC_HAL_GetConvPowerUpFlag (uint32_t *baseAddr*, cadc_conv_id_t *convId*)

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	ID for ADC converter, see to "cadc_conv_id_t".

Returns

Assertion of indicated event.

6.2.3.42 static void CADC_HAL_SetPowerUpDelayClk (*uint32_t baseAddr, uint16_t val*) [inline], [static]

This 6-bit setting value determines the number of ADC clocks provided to power up an ADC converter before allowing a scan to start. It also determines the number of ADC clocks of delay provided in auto-powerdown and auto-standby modes between when the ADC goes from the idle to active state and when the scan is allowed to start. The default value is 26 ADC clocks. Accuracy of the initial conversions in a scan will be degraded if the setting is set to too small a value.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>val</i>	6-bit setting value.

6.2.3.43 static uint16_t CADC_HAL_GetPowerUpDelayClk (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

6-bit setting value.

6.2.3.44 static void CADC_HAL_SetAutoPowerDownCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

Auto-powerdown mode powers down converters when not in use for a scan. Auto-powerdown (APD) takes precedence over auto-standby (ASB). When a scan is started in APD mode, a delay of setting ADC clock cycles is imposed during which the needed converter(s), if idle, are powered up. The ADC will then initiate a scan equivalent to that done when APD is not active. When the scan is completed, the converter(s) are powered down again.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	Assertion to enable the feature.

6.2.3.45 void CADC_HAL_SetConvPowerUpCmd (uint32_t *baseAddr*, *cadc_conv_id_t convId*, *bool enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	Selection of ID for ADC converter.
<i>enable</i>	Assertion to enable the feature.

6.2.3.46 void CADC_HAL_SetConvUseChnVrefHCmd (uint32_t *baseAddr*, *cadc_conv_id_t convId*, *bool enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	Selection of ID for ADC converter.
<i>enable</i>	Assertion to enable the feature.

6.2.3.47 void CADC_HAL_SetConvUseChnVrefLCmd (uint32_t *baseAddr*, *cadc_conv_id_t convId*, *bool enable*)

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	Selection of ID for ADC converter.
<i>enable</i>	Assertion to enable the feature.

6.2.3.48 void CADC_HAL_SetChnGainMode (uint32_t *baseAddr*, uint32_t *chnNum*, *cadc_gain_mode_t mode*)

CADC HAL Driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chnNum</i>	Channel number of input mux.
<i>mode</i>	Selection of gain mode, see to "cadc_gain_mode_t".

6.2.3.49 void CADC_HAL_SetSlotSyncPointCmd (uint32_t *baseAddr*, uint32_t *slotNum*, bool *enable*)

The sync point provides the ability to pause and await a new sync while processing samples programmed in conversion sequence. This API determines whether a sample in a scan occurs immediately or if the sample waits for an enabled sync input to occur. The sync input must occur after the conversion of the current sample completes.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>enable</i>	Assertion of indicated feature.

6.2.3.50 void CADC_HAL_SetConvSpeedLimitMode (uint32_t *baseAddr*, cadc_conv_id_t *convId*, cadc_conv_speed_mode_t *mode*)

Note, faster conversion speeds require greater current consumption.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	Selection of ID for ADC converter.
<i>mode</i>	Selection of speed mode, see to "cadc_conv_clk_limit_mode_t".

6.2.3.51 static void CADC_HAL_SetDmaTriggerSrcMode (uint32_t *baseAddr*, cadc_dma_trigger_src_mode_t *mode*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of setting mode, see to "cadc_dma_trigger_src_mode_t".

6.2.3.52 void CADC_HAL_SetConvSampleWindow (*uint32_t baseAddr, cadc_conv_id_t convId, uint16_t val*)

During sequential and parallel simultaneous scan modes, the setting value controls the sampling time of the first sample after a scan is initiated on both converters A and B. In parallel non-simultaneous mode, the setting value for each converter affects converter A or B only. The default value is 0 which corresponds to a sampling time of 2 ADC clocks. Each increment of the setting value corresponds to an additional ADC clock cycle of sampling time with a maximum sampling time of 9 ADC clocks. In sequential scan mode, the converter A's setting will be ignored whenever the channel selected for the next sample is on the other converter. In other words, during a sequential scan, if a sample converts a converter A channel (ANA0-ANA7) and the next sample converts a converter B channel (ANB0-ANB7) or vice versa, the setting value will be ignored and use the default sampling time for the next sample.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>convId</i>	Selection of ID for ADC converter.
<i>3-bit</i>	setting value.

6.2.3.53 void CADC_HAL_SetSlotScanIntCmd (*uint32_t baseAddr, uint32_t slotNum, bool enable*)

This API is used with ready flag detection to select the samples that will generate a scan interrupt.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>slotNum</i>	Slot number in conversion sequence.
<i>enable</i>	Assertion of indicated feature.

6.3 CADC Peripheral Driver

6.3.1 Overview

This section describes the programming interface of the CADC Peripheral driver. The CADC peripheral driver configures the Cyclic ADC (12-bit Cyclic Analog-to-Digital Converter). It handles initialization, configuration and controlling the conversion of this module.

6.3.2 CADC Driver model building

CADC driver has four objects:

- ADC Converter - CADC module has two ADC converters, ConvA and ConvB, which are the two independent sample and hold (S/H) circuits. They can work simultaneously or individually for each conversion. The driver provides APIs to configure each converter separately.
 - Conversion sequence - Conversion sequence is the basic execution unit for the CADC module. Starting a single conversion either launches a sequence of conversions or a scan. Each slot in the sequence can be configured as an input sample channel from the channel multiplexer. The conversion result is stored in the related result register. Additional features, such as zero-crossing detection and low/high limit detection can be attached to the conversion for each slot. The driver provides APIs to configure each slot in the sequence.
 - Input multiplexer - 16 input single-end inputs for the CADC module are grouped for each converter, ANA0 - ANA7 belong to ConvA, ANB0 - ANB7 belong to ConvB. As a result, only the unique host converter can convert the indicated input channel. Inputs are grouped for the differential sample so that ANA[0:1], ANA[2:3], ANA[4:5], ANA[6:7], ANB[0:1], ANB[2:3], ANB[4:5], ANB[6:7]. In this case, a channel with an even number is positive and the channel with an odd number is negative. The driver provides APIs to configure each input channel.
 - Global setting - Global setting covers all shared configurations. The driver provides APIs to configure the global setting.

6.3.3 CADC Work mode

The ADC operates either in a sequential scan mode or a parallel scan mode. In the sequential scan mode, the scan sequence is determined by defining the 16 sample slots that are processed in order, SAMPLE[0:15]. In the parallel scan mode, ConvA processes SAMPLE[0:7] in order and ConvB processes SAMPLE[8:15] in order.

- In the sequential scan mode, a scan takes up to 16 single-ended or differential samples, one at a time.
- In the parallel scan mode, 8 of the 16 samples are allocated to ConvA and the other 8 are allocated to ConvB. Two converters operate in parallel such that each can take at most 8 samples. ConvA can sample only analog inputs ANA[0:7] and ConvB can sample only analog inputs ANB[0:7].

The parallel scan mode is simultaneous or non-simultaneous:

- In the simultaneous scan mode, parallel scans in the two converters occur simultaneously and result in simultaneous pairs of conversions, one by ConvA and one by ConvB. The two converters share the same start, stop, sync, end-of-scan interrupt enable control, and interrupts. Scanning in both converters terminates when either converter encounters a disabled sample.
- In the non-simultaneous scan mode, parallel scans in the two converters occur independently. Each converter has its own start, stop, sync, end-of-scan interrupt enable controls, and interrupts. Scanning in either converter terminates only when that converter encounters a disabled sample.

The ADC can be configured to perform a single scan and halt, perform a scan whenever triggered, or perform the scan sequence repeatedly until manually stopped. The single scan (once mode) differs from the triggered mode only in that SYNC input signals must be re-armed after each use and subsequent SYNC inputs are ignored until the SYNC input is re-armed. Arming can occur any time after the SYNC pulse, including while the scan is still in process.

6.3.4 CADC Interrupt

There are three categories of ADC interrupts:

- Threshold interrupts, which are caused by three different events:
 - Zero crossing — occurs if the current result value has a sign change from the previous result.
 - Low limit exceeded error — occurs when the current result value is less than the setting value for the low limit. The raw result value is compared to the setting value for the low limit before the offset register value is subtracted.
 - High limit exceeded error — is asserted if the current result value is greater than the setting value for the high limit. The raw result value is compared to the setting value for the high limit.
- Conversion complete interrupts, which are generated upon completion of any scan and convert sequence when enabling the interrupt.
- Scan interrupts are generated when a sample is converted. This allows processing of intermediate conversion data during a scan.

6.3.5 CADC Call diagram

When using the CADC module, all four objects should be configured according to the application requirements. This example illustrates how to use the CADC driver.

```
// Four kinds of structures to configure Cyclic ADC with conversion sequence. //
cadc_user_config_t g_Ad_usrConfigStruct;
cadc_chn_config_t g_Ad_usrChnConfigStruct;
cadc_conv_config_t g_Ad_usrConvConfigStruct;
cadc_slot_config_t g_Ad_usrSlotConfigStruct;

static void CADC_DRV_TEST_PollingMode(uint32_t instance)
{
    volatile int tmp;

    CADC_DRV_StructInitUserConfigDefault(&g_Ad_usrConfigStruct);
    CADC_DRV_Init(instance, &g_Ad_usrConfigStruct);
```

CADC Peripheral Driver

```
// Configure ADC sample input channel. //
g_AdcChnConfigStruct.diffEnable = true;
g_AdcChnConfigStruct.diffSelMode = kCAdcChnSelBoth;
g_AdcChnConfigStruct.gainMode = kCAdcSGainBy1;
CADC_DRV_ConfigSampleChn(instance,
    kCAdcDiffChnANA4_5, &g_AdcChnConfigStruct);
CADC_DRV_ConfigSampleChn(instance,
    kCAdcDiffChnANA6_7, &g_AdcChnConfigStruct);
CADC_DRV_ConfigSampleChn(instance,
    kCAdcDiffChnANB4_5, &g_AdcChnConfigStruct);
CADC_DRV_ConfigSampleChn(instance,
    kCAdcDiffChnANB6_7, &g_AdcChnConfigStruct);

// Configure ADC converters. //
g_AdcConvConfigStruct.dmaEnable = false;
g_AdcConvConfigStruct.stopEnable = false; // Ungate the converter. //
g_AdcConvConfigStruct.syncEnable = false; // Software trigger only. //
g_AdcConvConfigStruct.endOfScanIntEnable = false; // No interrupt. //
g_AdcConvConfigStruct.convIRQEnable = false;
g_AdcConvConfigStruct.clkDivValue = 0x4U;
g_AdcConvConfigStruct.powerOnEnable = true; // Power on the converter. //
g_AdcConvConfigStruct.useChnInputAsVrefH = false;
g_AdcConvConfigStruct.useChnInputAsVrefL = false;
g_AdcConvConfigStruct.speedMode = kCAdcConvClkLimitBy25MHz;
g_AdcConvConfigStruct.sampleWindowCount = 0U;
CADC_DRV_ConfigConverter(instance, kCAdcConvA, &g_AdcConvConfigStruct
);
CADC_DRV_ConfigConverter(instance, kCAdcConvB, &g_AdcConvConfigStruct
);
while (!CADC_DRV_GetConvFlag(instance, kCAdcConvA,
    kCAdcConvPowerUp) ) {}
while (!CADC_DRV_GetConvFlag(instance, kCAdcConvB,
    kCAdcConvPowerUp) ) {}

// Configure slot in conversion sequence. //
// Common setting. //
g_AdcSlotConfigStruct.zeroCrossingMode =
    kCAdcZeroCrossingDisable;
g_AdcSlotConfigStruct.lowLimitValue = 0U;
g_AdcSlotConfigStruct.highLimitValue = 0xFFFFU;
g_AdcSlotConfigStruct.offsetValue = 0U;
g_AdcSlotConfigStruct.syncPointEnable = false;
g_AdcSlotConfigStruct.scanIntEnable = false;
// For each slot in conversion sequence. //
// Slot 0. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANA4_5;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotEnable = true;
CADC_DRV_ConfigSeqSlot(instance, 0U, &g_AdcSlotConfigStruct);
// Slot 1. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANA6_7;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotEnable = true;
CADC_DRV_ConfigSeqSlot(instance, 1U, &g_AdcSlotConfigStruct);
// Slot 2. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANB4_5;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotEnable = true;
CADC_DRV_ConfigSeqSlot(instance, 2U, &g_AdcSlotConfigStruct);
// Slot 3. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANB6_7;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotEnable = true;
CADC_DRV_ConfigSeqSlot(instance, 3U, &g_AdcSlotConfigStruct);
g_AdcSlotConfigStruct.slotEnable = false;
CADC_DRV_ConfigSeqSlot(instance, 4U, &g_AdcSlotConfigStruct);
```

```

// Trigger the conversion sequence.
// When in sequential simult mode, operate converter A will driver the
// main conversion sequence.
//
CADC_DRV_SoftTriggerConv(instance, kCAdcConvA);

// Wait the data to be ready. //
while (!CADC_DRV_GetSlotFlag(instance, 0U,
    kCAdcSlotReady)) { }
while (!CADC_DRV_GetSlotFlag(instance, 1U,
    kCAdcSlotReady)) { }
while (!CADC_DRV_GetSlotFlag(instance, 2U,
    kCAdcSlotReady)) { }
while (!CADC_DRV_GetSlotFlag(instance, 3U,
    kCAdcSlotReady)) { }

tmp = CADC_DRV_GetSeqSlotConvValueSigned(instance, 0U);
printf("ADC Slot %2d value: %d\r\n", 0U, tmp);
tmp = CADC_DRV_GetSeqSlotConvValueSigned(instance, 1U);
printf("ADC Slot %2d value: %d\r\n", 1U, tmp);
tmp = CADC_DRV_GetSeqSlotConvValueSigned(instance, 2U);
printf("ADC Slot %2d value: %d\r\n", 2U, tmp);
tmp = CADC_DRV_GetSeqSlotConvValueSigned(instance, 3U);
printf("ADC Slot %2d value: %d\r\n", 3U, tmp);

CADC_DRV_Deinit(instance);
}

```

Data Structures

- struct [cadc_user_config_t](#)
Defines a structure to configure the CyclicADC module during initialization. [More...](#)
- struct [cadc_conv_config_t](#)
Defines a structure to configure each converter in the CyclicADC module. [More...](#)
- struct [cadc_chn_config_t](#)
Defines a structure to configure each input channel. [More...](#)
- struct [cadc_slot_config_t](#)
Defines a structure to configure each slot. [More...](#)

Enumerations

- enum [cadc_flag_t](#) {
 kCAdcConvInProgress = 0U,
 kCAdcConvEndOfScanInt = 1U,
 kCAdcConvPowerUp = 2U,
 kCAdcZeroCrossingInt = 3U,
 kCAdcLowLimitInt = 4U,
 kCAdcHighLimitInt = 5U,
 kCAdcSlotReady = 6U,
 kCAdcSlotLowLimitEvent = 7U,
 kCAdcSlotHighLimitEvent = 8U,
 kCAdcSlotCrossingEvent = 9U }
- Defines types for an enumerating event.*

CADC Peripheral Driver

Variables

- const uint32_t **g_cadcBaseAddr** []
Table of base addresses for ADC instances.
- const IRQn_Type **g_cadcErrIrqId** [HW_ADC_INSTANCE_COUNT]
Table to save ADC IRQ enumeration numbers defined in the CMSIS header file.

CADC Driver

- **cadc_status_t CADC_DRV_StructInitUserConfigDefault** (**cadc_user_config_t** *configPtr)
Populates the user configuration structure for the CyclicADC common settings.
- **cadc_status_t CADC_DRV_Init** (uint32_t instance, **cadc_user_config_t** *configPtr)
Initializes the CyclicADC module for a global configuration.
- void **CADC_DRV_Deinit** (uint32_t instance)
De-initializes the CyclicADC module.
- **cadc_status_t CADC_DRV_StructInitConvConfigDefault** (**cadc_conv_config_t** *configPtr)
Populates the user configuration structure for each converter.
- **cadc_status_t CADC_DRV_ConfigConverter** (uint32_t instance, **cadc_conv_id_t** convId, **cadc_conv_config_t** *configPtr)
Configures each converter in the CyclicADC module.
- **cadc_status_t CADC_DRV_ConfigSampleChn** (uint32_t instance, **cadc_diff_chn_mode_t** diffChns, **cadc_chn_config_t** *configPtr)
Configures the input channel for ADC conversion.
- **cadc_status_t CADC_DRV_ConfigSeqSlot** (uint32_t instance, uint32_t slotNum, **cadc_slot_config_t** *configPtr)
Configures each slot for the ADC conversion sequence.
- void **CADC_DRV_SoftTriggerConv** (uint32_t instance, **cadc_conv_id_t** convId)
Triggers the ADC conversion sequence by software.
- uint16_t **CADC_DRV_GetSeqSlotConvValueRAW** (uint32_t instance, uint32_t slotNum)
Reads the conversion value and returns an absolute value.
- int16_t **CADC_DRV_GetSeqSlotConvValueSigned** (uint32_t instance, uint32_t slotNum)
Reads the conversion value and returns a signed value.
- bool **CADC_DRV_GetFlag** (uint32_t instance, **cadc_flag_t** flag)
Gets the global event flag.
- void **CADC_DRV_ClearFlag** (uint32_t instance, **cadc_flag_t** flag)
Clears the global event flag.
- bool **CADC_DRV_GetConvFlag** (uint32_t instance, **cadc_conv_id_t** convId, **cadc_flag_t** flag)
Gets the flag for each converter event.
- void **CADC_DRV_ClearConvFlag** (uint32_t instance, **cadc_conv_id_t** convId, **cadc_flag_t** flag)
Clears the flag for each converter event.
- bool **CADC_DRV_GetSlotFlag** (uint32_t instance, uint32_t slotNum, **cadc_flag_t** flag)
Gets the flag for each slot event.
- void **CADC_DRV_ClearSlotFlag** (uint32_t instance, uint32_t slotNum, **cadc_flag_t** flag)
Clears the flag for each slot event.

6.3.6 Data Structure Documentation

6.3.6.1 struct `cadc_user_config_t`

This structure holds the configuration when initializing the CyclicADC module.

Data Fields

- bool `zeroCrossingIntEnable`
Global zero crossing interrupt enable.
- bool `lowLimitIntEnable`
Global low limit interrupt enable.
- bool `highLimitIntEnable`
Global high limit interrupt enable.
- `cadc_scan_mode_t scanMode`
ADC scan mode control.
- bool `parallelSimultModeEnable`
Parallel scans done simultaneously enable.
- `cadc_dma_trigger_src_mode_t dmaSrcMode`
DMA trigger source.
- bool `autoStandbyEnable`
Auto standby mode enable.
- uint16_t `powerUpDelayCount`
Power up delay.
- bool `autoPowerDownEnable`
Auto power down mode enable.

6.3.6.1.0.7 Field Documentation

6.3.6.1.0.7.1 bool `cadc_user_config_t::zeroCrossingIntEnable`

6.3.6.1.0.7.2 bool `cadc_user_config_t::lowLimitIntEnable`

6.3.6.1.0.7.3 bool `cadc_user_config_t::highLimitIntEnable`

6.3.6.1.0.7.4 `cadc_scan_mode_t` `cadc_user_config_t::scanMode`

See "cadc_scan_mode_t".

6.3.6.1.0.7.5 bool `cadc_user_config_t::parallelSimultModeEnable`

6.3.6.1.0.7.6 `cadc_dma_trigger_src_mode_t` `cadc_user_config_t::dmaSrcMode`

See "cadc_dma_trigger_src_mode_t".

CADC Peripheral Driver

6.3.6.1.0.7.7 `bool cadc_user_config_t::autoStandbyEnable`

6.3.6.1.0.7.8 `uint16_t cadc_user_config_t::powerUpDelayCount`

6.3.6.1.0.7.9 `bool cadc_user_config_t::autoPowerDownEnable`

6.3.6.2 `struct cadc_conv_config_t`

This structure holds the configuration for each converter in the CyclicADC module. Normally, there are two converters, ConvA and ConvB in the cyclic ADC module. However, each converter can be configured separately for some features.

Data Fields

- `bool dmaEnable`
DMA enable.
- `bool stopEnable`
Stop mode enable.
- `bool syncEnable`
Enable external sync input to trigger conversion.
- `bool endOfScanIntEnable`
End of scan interrupt enable.
- `bool convIRQEnable`
Enable IRQ in NVIC; Cover the slot interrupt in the following configuration.
- `uint16_t clkDivValue`
ADC clock divider from the bus clock.
- `bool powerOnEnable`
Disable manual power down for converter.
- `bool useChnInputAsVrefH`
Use input channel as high reference voltage, such as AN2.
- `bool useChnInputAsVrefL`
Use input channel as low reference voltage, such as AN3.
- `cadc_conv_speed_mode_t speedMode`
ADC speed control mode, see "cadc_conv_speed_mode_t".
- `uint16_t sampleWindowCount`
Sample window count.

6.3.6.2.0.8 Field Documentation

- 6.3.6.2.0.8.1 **bool** `cadc_conv_config_t::dmaEnable`
- 6.3.6.2.0.8.2 **bool** `cadc_conv_config_t::stopEnable`
- 6.3.6.2.0.8.3 **bool** `cadc_conv_config_t::syncEnable`
- 6.3.6.2.0.8.4 **bool** `cadc_conv_config_t::endOfScanIntEnable`
- 6.3.6.2.0.8.5 **bool** `cadc_conv_config_t::convIRQEnable`
- 6.3.6.2.0.8.6 **uint16_t** `cadc_conv_config_t::clkDivValue`
- 6.3.6.2.0.8.7 **bool** `cadc_conv_config_t::powerOnEnable`
- 6.3.6.2.0.8.8 **bool** `cadc_conv_config_t::useChnInputAsVrefH`
- 6.3.6.2.0.8.9 **bool** `cadc_conv_config_t::useChnInputAsVrefL`
- 6.3.6.2.0.8.10 **cadc_conv_speed_mode_t** `cadc_conv_config_t::speedMode`
- 6.3.6.2.0.8.11 **uint16_t** `cadc_conv_config_t::sampleWindowCount`

6.3.6.3 **struct** `cadc_chn_config_t`

This structure holds the configuration for each input channel. In CyclicADC module, the input channels are handled by a differential sample. However, the user can still configure the function for each channel when set to operate as a single end sample.

Data Fields

- **cadc_chn_sel_mode_t** `diffSelMode`
Select which channel is indicated in a pair.

6.3.6.3.0.9 Field Documentation

- 6.3.6.3.0.9.1 **cadc_chn_sel_mode_t** `cadc_chn_config_t::diffSelMode`

6.3.6.4 **struct** `cadc_slot_config_t`

This structure holds the configuration for each slot in a conversion sequence.

Data Fields

- **bool** `syncPointEnable`
Sample waits for an enabled SYNC input to occur.
- **bool** `scanIntEnable`
Scan interrupt enable.

CADC Peripheral Driver

- `cadc_diff_chn_mode_t diffChns`
Select the differential pair.
- `cadc_chn_sel_mode_t diffSel`
Positive or negative channel in differential pair.
- `cadc_zero_crossing_mode_t zeroCrossingMode`
Select zero crossing detection mode.
- `uint16_t lowLimitValue`
Select low limit for hardware compare.
- `uint16_t highLimitValue`
Select high limit for hardware compare.
- `uint16_t offsetValue`
Select sign change limit for hardware compare.

6.3.6.4.0.10 Field Documentation

6.3.6.4.0.10.1 `bool cadc_slot_config_t::syncPointEnable`

6.3.6.4.0.10.2 `bool cadc_slot_config_t::scanIntEnable`

6.3.6.4.0.10.3 `cadc_diff_chn_mode_t cadc_slot_config_t::diffChns`

6.3.6.4.0.10.4 `cadc_chn_sel_mode_t cadc_slot_config_t::diffSel`

6.3.6.4.0.10.5 `cadc_zero_crossing_mode_t cadc_slot_config_t::zeroCrossingMode`

6.3.6.4.0.10.6 `uint16_t cadc_slot_config_t::lowLimitValue`

6.3.6.4.0.10.7 `uint16_t cadc_slot_config_t::highLimitValue`

6.3.6.4.0.10.8 `uint16_t cadc_slot_config_t::offsetValue`

6.3.7 Enumeration Type Documentation

6.3.7.1 `enum cadc_flag_t`

Enumerator

kCAdcConvInProgress Conversion in progress for each converter.

kCAdcConvEndOfScanInt End of scan interrupt.

kCAdcConvPowerUp The converter is powered up.

kCAdcZeroCrossingInt Zero crossing interrupt.

kCAdcLowLimitInt Low limit interrupt.

kCAdcHighLimitInt High limit interrupt.

kCAdcSlotReady Sample is ready to be read.

kCAdcSlotLowLimitEvent Low limit event for each slot.

kCAdcSlotHighLimitEvent High limit event for each slot.

kCAdcSlotCrossingEvent Zero crossing event for each slot.

6.3.8 Function Documentation

6.3.8.1 `cadc_status_t CADC_DRV_StructInitUserConfigDefault (cadc_user_config_t * configPtr)`

This function populates the `cadc_user_config_t` structure with default settings, which are used in polling mode for ADC conversion. These settings are:

```
.zeroCrossingIntEnable = false; .lowLimitIntEnable = false; .highLimitIntEnable = false; .scanMode = kCAdcScanOnceSequential; .parallelSimultModeEnable = false; .dmaSrcMode = kCAdcDmaTriggeredByEndOfScan; .autoStandbyEnable = false; .powerUpDelayCount = 0x2AU; .autoPowerDownEnable = false;
```

Parameters

<i>configPtr</i>	Pointer to structure of "cadc_user_config_t".
------------------	---

Returns

Execution status.

6.3.8.2 `cadc_status_t CADC_DRV_Init (uint32_t instance, cadc_user_config_t * configPtr)`

This function configures the CyclicADC module for the global configuration which is shared by all converters.

Parameters

<i>instance</i>	Instance ID number.
<i>configPtr</i>	Pointer to structure of "cadc_user_config_t".

Returns

Execution status.

6.3.8.3 `void CADC_DRV_Deinit (uint32_t instance)`

This function shuts down the CyclicADC module and disables related IRQ.

CADC Peripheral Driver

Parameters

<i>instance</i>	Instance ID number.
-----------------	---------------------

6.3.8.4 **cadc_status_t CADC_DRV_StructInitConvConfigDefault (*cadc_conv_config_t* * *configPtr*)**

This function populates the *cadc_conv_config_t* structure with default settings, which are used in polling mode for ADC conversion. These settings are:

```
.dmaEnable = false; .stopEnable = false; .syncEnable = false; .endOfScanIntEnable = false; .convIRQEnable = false; .clkDivValue = 0x3FU; .powerOnEnable = true; .useChnInputAsVrefH = false; .useChnInputAsVrefL = false; .speedMode = kCAdcConvClkLimitBy25MHz; .sampleWindowCount = 0U;
```

Parameters

<i>configPtr</i>	Pointer to structure of "cadc_conv_config_t".
------------------	---

Returns

Execution status.

6.3.8.5 **cadc_status_t CADC_DRV_ConfigConverter (*uint32_t instance*, *cadc_conv_id_t convId*, *cadc_conv_config_t* * *configPtr*)**

This function configures each converter in the CyclicADC module. However, when the multiple converters are operating simultaneously, the converter settings are interrelated. For more information, see the appropriate device reference manual.

Parameters

<i>instance</i>	Instance ID number.
<i>convId</i>	Converter ID. See "cadc_conv_id_t".
<i>configPtr</i>	Pointer to configure structure. See "cadc_conv_config_t".

Returns

Execution status.

6.3.8.6 **cadc_status_t CADC_DRV_ConfigSampleChn (*uint32_t instance*, *cadc_diff_chn_mode_t diffChns*, *cadc_chn_config_t* * *configPtr*)**

This function configures the input channel for ADC conversion. The CyclicADC module input channels are organized in pairs. The configuration can be set for each channel in the pair.

Parameters

<i>instance</i>	Instance ID number.
<i>diffChns</i>	Differential channel pair. See "cadc_diff_chn_mode_t".
<i>configPtr</i>	Pointer to configure structure. See "cadc_chn_config_t".

Returns

Execution status.

6.3.8.7 **cadc_status_t CADC_DRV_ConfigSeqSlot (uint32_t *instance*, uint32_t *slotNum*, cadc_slot_config_t * *configPtr*)**

This function configures each slot in the ADC conversion sequence. ADC conversion sequence is the basic execution unit in the CyclicADC module. However, the sequence should be configured slot-by-slot. The end of the sequence is a slot that is configured as disabled.

Parameters

<i>instance</i>	Instance ID number.
<i>slotNum</i>	Indicated slot number, available in range of 0 - 15.
<i>configPtr</i>	Pointer to configure structure. See "cadc_slot_config_t".

Returns

Execution status.

6.3.8.8 **void CADC_DRV_SoftTriggerConv (uint32_t *instance*, cadc_conv_id_t *convId*)**

This function triggers the ADC conversion by executing a software command. It starts the conversion if no other SYNC input (hardware trigger) is needed.

Parameters

<i>instance</i>	Instance ID number.
<i>convId</i>	Indicated converter. See "cadc_conv_id_t".

6.3.8.9 **uint16_t CADC_DRV_GetSeqSlotConvValueRAW (uint32_t *instance*, uint32_t *slotNum*)**

This function reads the conversion value from each slot in a conversion sequence. The return value is the absolute value without being signed.

CADC Peripheral Driver

Parameters

<i>instance</i>	Instance ID number.
<i>slotNum</i>	Indicated slot number, available in range of 0 - 15.

6.3.8.10 int16_t CADC_DRV_GetSeqSlotConvValueSigned (uint32_t *instance*, uint32_t *slotNum*)

This function reads the conversion value from each slot in the conversion sequence. The return value is a signed value. When the read value is negative, the sample value is lower than the offset value.

Parameters

<i>instance</i>	Instance ID number.
<i>slotNum</i>	Indicated slot number, available in range of 0 - 15.

6.3.8.11 bool CADC_DRV_GetFlag (uint32_t *instance*, cadc_flag_t *flag*)

This function gets the global flag of the CyclicADC module.

Parameters

<i>instance</i>	Instance ID number.
<i>flag</i>	Indicated event. See "cadc_flag_t".

Returns

Assertion of indicated event.

6.3.8.12 void CADC_DRV_ClearFlag (uint32_t *instance*, cadc_flag_t *flag*)

This function clears the global event flag of the CyclicADC module.

Parameters

<i>instance</i>	Instance ID number.
-----------------	---------------------

<i>flag</i>	Indicated event. See "cadc_flag_t".
-------------	-------------------------------------

6.3.8.13 **bool CADC_DRV_GetConvFlag (uint32_t *instance*, cadc_conv_id_t *convId*, cadc_flag_t *flag*)**

This function gets the flag for each converter event.

Parameters

<i>instance</i>	Instance ID number.
<i>convId</i>	Indicated converter.
<i>flag</i>	Indicated event. See "cadc_flag_t".

Returns

Assertion of indicated event.

6.3.8.14 **void CADC_DRV_ClearConvFlag (uint32_t *instance*, cadc_conv_id_t *convId*, cadc_flag_t *flag*)**

This function clears the flag for each converter event.

Parameters

<i>instance</i>	Instance ID number.
<i>convId</i>	Indicated converter.
<i>flag</i>	Indicated event. See "cadc_flag_t".

6.3.8.15 **bool CADC_DRV_GetSlotFlag (uint32_t *instance*, uint32_t *slotNum*, cadc_flag_t *flag*)**

This function gets the flag for each slot event in the conversion in sequence.

Parameters

<i>instance</i>	Instance ID number.
<i>slotNum</i>	Indicated slot number, available in range of 0 - 15.
<i>flag</i>	Indicated event. See "cadc_flag_t".

CADC Peripheral Driver

Returns

Assertion of indicated event.

6.3.8.16 void CADC_DRV_ClearSlotFlag (*uint32_t instance*, *uint32_t slotNum*, *cadc_flag_t flag*)

This function clears the flag for each slot event in the conversion in sequence.

Parameters

<i>instance</i>	Instance ID number.
<i>slotNum</i>	Indicated slot number, available in range of 0 - 15.
<i>flag</i>	Indicated event. See "cadc_flag_t".

6.3.9 Variable Documentation

6.3.9.1 const uint32_t g_cadcBaseAddr[]

6.3.9.2 const IRQn_Type g_cadcErrIrqlId[HW_ADC_INSTANCE_COUNT]

Chapter 7

Digital-to-Analog Converter (DAC)

7.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Digital-to-Analog Converter block of Kinetis devices.

Modules

- [DAC HAL Driver](#)
- [DAC Peripheral Driver](#)

7.2 DAC HAL Driver

7.2.1 Overview

This section describes the programming interface of the DAC HAL driver.

Enumerations

- enum `dac_status_t` {
 `kStatus_DAC_Success` = 0U,
 `kStatus_DAC_InvalidArgument` = 1U,
 `kStatus_DAC_Failed` = 2U }
 DAC status return codes.
- enum `dac_ref_volt_src_mode_t` {
 `kDacRefVoltSrcOfVref1` = 0U,
 `kDacRefVoltSrcOfVref2` = 1U }
 Defines the type of selection for DAC module's reference voltage source.
- enum `dac_trigger_mode_t` {
 `kDacTriggerByHardware` = 0U,
 `kDacTriggerBySoftware` = 1U }
 Defines the type of selection for DAC module trigger mode.
- enum `dac_buff_watermark_mode_t` {
 `kDacBuffWatermarkFromUpperAs1Word` = 0U,
 `kDacBuffWatermarkFromUpperAs2Word` = 1U,
 `kDacBuffWatermarkFromUpperAs3Word` = 2U,
 `kDacBuffWatermarkFromUpperAs4Word` = 3U }
 Defines the type of selection for buffer watermark mode.
- enum `dac_buff_work_mode_t` { `kDacBuffWorkAsNormalMode` = 0U }
 Defines the type of selection for buffer work mode.

Functions

- void `DAC_HAL_Init` (uint32_t baseAddr)
 Resets all configurable registers to be in the reset state for DAC.
- void `DAC_HAL_SetBuffValue` (uint32_t baseAddr, uint8_t index, uint16_t value)
 Sets the 12-bit value for the DAC items in the buffer.
- uint16_t `DAC_HAL_GetBuffValue` (uint32_t baseAddr, uint8_t index)
 Gets the 12-bit value from the DAC item in the buffer.
- static void `DAC_HAL_ClearBuffIndexUpperFlag` (uint32_t baseAddr)
 Clears the flag of the DAC buffer read pointer.
- static bool `DAC_HAL_GetBuffIndexUpperFlag` (uint32_t baseAddr)
 Gets the flag of DAC buffer read pointer when it hits the bottom position.
- static void `DAC_HAL_ClearBuffIndexStartFlag` (uint32_t baseAddr)
 Clears the flag of the DAC buffer read pointer when it hits the top position.
- static bool `DAC_HAL_GetBuffIndexStartFlag` (uint32_t baseAddr)
 Gets the flag of the DAC buffer read pointer when it hits the top position.
- static void `DAC_HAL_Enable` (uint32_t baseAddr)

- static void **DAC_HAL_Disable** (uint32_t baseAddr)

Enables the Programmable Reference Generator.
- static void **DAC_HAL_SetRefVoltSrcMode** (uint32_t baseAddr, **dac_ref_volt_src_mode_t** mode)

Disables the Programmable Reference Generator.

Sets the reference voltage source mode for the DAC module.
- static void **DAC_HAL_SetTriggerMode** (uint32_t baseAddr, **dac_trigger_mode_t** mode)

Sets the trigger mode for the DAC module.
- static void **DAC_HAL_SetSoftTriggerCmd** (uint32_t baseAddr)

Triggers the converter with software.
- static void **DAC_HAL_SetLowPowerCmd** (uint32_t baseAddr, bool enable)

Switches to enable working in low power mode for the DAC module.
- static void **DAC_HAL_SetBuffIndexStartIntCmd** (uint32_t baseAddr, bool enable)

Switches to enable the interrupt when the buffer read pointer hits the top position.
- static void **DAC_HAL_SetBuffIndexUpperIntCmd** (uint32_t baseAddr, bool enable)

Switches to enable the interrupt when the buffer read pointer hits the bottom position.
- static void **DAC_HAL_SetDmaCmd** (uint32_t baseAddr, bool enable)

Switches to enable the DMA for DAC.
- static void **DAC_HAL_SetBuffWorkMode** (uint32_t baseAddr, **dac_buff_work_mode_t** mode)

Sets the work mode of the buffer for the DAC module.
- static void **DAC_HAL_SetBuffCmd** (uint32_t baseAddr, bool enable)

Switches to enable the buffer for the DAC module.
- static uint8_t **DAC_HAL_GetBuffUpperIndex** (uint32_t baseAddr)

Gets the buffer index upper limitation for the DAC module.
- static void **DAC_HAL_SetBuffUpperIndex** (uint32_t baseAddr, uint8_t index)

Sets the buffer index upper limitation for the DAC module.
- static uint8_t **DAC_HAL_GetBuffCurrentIndex** (uint32_t baseAddr)

Gets the current buffer index upper limitation for the DAC module.
- static void **DAC_HAL_SetBuffcurrentIndex** (uint32_t baseAddr, uint8_t index)

Sets the buffer index for the DAC module.

7.2.2 Enumeration Type Documentation

7.2.2.1 enum dac_status_t

Enumerator

kStatus_DAC_Success Success.

kStatus_DAC_InvalidArgument Invalid argument existed.

kStatus_DAC_Failed Execution failed.

7.2.2.2 enum dac_ref_volt_src_mode_t

See the appropriate SoC Reference Manual for actual connections.

Enumerator

kDacRefVoltSrcOfVref1 Select DACREF_1 as the reference voltage.

kDacRefVoltSrcOfVref2 Select DACREF_2 as the reference voltage.

DAC HAL Driver

7.2.2.3 enum dac_trigger_mode_t

Enumerator

kDacTriggerByHardware Select hardware trigger.

kDacTriggerBySoftware Select software trigger.

7.2.2.4 enum dac_buff_watermark_mode_t

If the buffer feature for DAC module is enabled, a watermark event will occur when the buffer index hits the watermark.

Enumerator

kDacBuffWatermarkFromUpperAs1Word Select 1 word away from the upper of buffer.

kDacBuffWatermarkFromUpperAs2Word Select 2 word away from the upper of buffer.

kDacBuffWatermarkFromUpperAs3Word Select 3 word away from the upper of buffer.

kDacBuffWatermarkFromUpperAs4Word Select 4 word away from the upper of buffer.

7.2.2.5 enum dac_buff_work_mode_t

These are the work modes when the DAC buffer is enabled.

- Normal mode - When the buffer index hits the upper level, it starts (0) on the next trigger.
- Swing mode - When the buffer index hits the upper level, it goes backward to the start and is reduced one-by-one on the next trigger. When the buffer index hits the start, it goes backward to the upper level and increases one-by-one on the next trigger.
- One-Time-Scan mode - The buffer index can only be increased on the next trigger. When the buffer index hits the upper level, it is not updated by the trigger.
- FIFO mode - In FIFO mode, the buffer is organized as a FIFO. For a valid write to any item, the data will be put into the FIFO. The written index in buffer should be an EVEN number; otherwise, the write will be ignored.

Enumerator

kDacBuffWorkAsNormalMode Buffer works as Normal.

7.2.3 Function Documentation

7.2.3.1 void DAC_HAL_Init(uint32_t baseAddr)

This function resets all configurable registers to be in the reset state for DAC. It should be called before configuring the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

7.2.3.2 void DAC_HAL_SetBuffValue (uint32_t *baseAddr*, uint8_t *index*, uint16_t *value*)

This function sets the value assembled by the low 8 bits and high 4 bits of 12-bit DAC item in the buffer.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>index</i>	Buffer index.
<i>value</i>	Setting value.

7.2.3.3 uint16_t DAC_HAL_GetBuffValue (uint32_t *baseAddr*, uint8_t *index*)

This function gets the value assembled by the low 8 bits and high 4 bits of 12-bit DAC item in the buffer.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>index</i>	Buffer index.

Returns

Current setting value.

7.2.3.4 static void DAC_HAL_ClearBuffIndexUpperFlag (uint32_t *baseAddr*) [inline], [static]

This function clears the flag of the DAC buffer read pointer when it hits the bottom position.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

7.2.3.5 static bool DAC_HAL_GetBuffIndexUpperFlag (uint32_t *baseAddr*) [inline], [static]

This function gets the flag of DAC buffer read pointer when it hits the bottom position.

DAC HAL Driver

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

Returns

Assertion of indicated event.

7.2.3.6 static void DAC_HAL_ClearBuffIndexStartFlag (uint32_t *baseAddr*) [inline], [static]

This function clears the flag of the DAC buffer read pointer when it hits the top position.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

7.2.3.7 static bool DAC_HAL_GetBuffIndexStartFlag (uint32_t *baseAddr*) [inline], [static]

This function gets the flag of the DAC buffer read pointer when it hits the top position.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

Returns

Assertion of indicated event.

7.2.3.8 static void DAC_HAL_Enable (uint32_t *baseAddr*) [inline], [static]

This function enables the Programmable Reference Generator. Then the DAC system is enabled.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

7.2.3.9 static void DAC_HAL_Disable (uint32_t *baseAddr*) [inline], [static]

This function disables the Programmable Reference Generator. Then the DAC system is disabled.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

7.2.3.10 static void DAC_HAL_SetRefVoltSrcMode (*uint32_t baseAddr*, *dac_ref_volt_src_mode_t mode*) [inline], [static]

This function sets the reference voltage source mode for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>mode</i>	Selection of enumeration mode. See to "dac_ref_volt_src_mode_t".

7.2.3.11 static void DAC_HAL_SetTriggerMode (*uint32_t baseAddr*, *dac_trigger_mode_t mode*) [inline], [static]

This function sets the trigger mode for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>mode</i>	Selection of enumeration mode. See to "dac_trigger_mode_t".

7.2.3.12 static void DAC_HAL_SetSoftTriggerCmd (*uint32_t baseAddr*) [inline], [static]

This function triggers the converter with software. If the DAC software trigger is selected and buffer enabled, calling this API advances the buffer read pointer once.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

7.2.3.13 static void DAC_HAL_SetLowPowerCmd (*uint32_t baseAddr*, *bool enable*) [inline], [static]

This function switches to enable working in low power mode for the DAC module.

DAC HAL Driver

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>enable</i>	Switcher to assert the feature.

7.2.3.14 static void DAC_HAL_SetBuffIndexStartIntCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the interrupt when the buffer read pointer hits the top position.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>enable</i>	Switcher to assert the feature.

7.2.3.15 static void DAC_HAL_SetBuffIndexUpperIntCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the interrupt when the buffer read pointer hits the bottom position.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>enable</i>	Switcher to assert the feature.

7.2.3.16 static void DAC_HAL_SetDmaCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function switches to enable the DMA for the DAC module. When the DMA is enabled, DMA request is generated by the original interrupts, which are not presented on this module at the same time.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>enable</i>	Switcher to assert the feature.

7.2.3.17 static void DAC_HAL_SetBuffWorkMode (*uint32_t baseAddr, dac_buff_work_mode_t mode*) [inline], [static]

This function sets the work mode of the buffer for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>mode</i>	Selection of enumeration mode. See to "dac_buff_work_mode_t".

7.2.3.18 static void DAC_HAL_SetBuffCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the buffer for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
<i>enable</i>	Switcher to assert the feature.

7.2.3.19 static uint8_t DAC_HAL_GetBuffUpperIndex (uint32_t *baseAddr*) [inline], [static]

This function gets the upper buffer index upper limitation for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

Returns

Value of buffer index upper limitation.

7.2.3.20 static void DAC_HAL_SetBuffUpperIndex (uint32_t *baseAddr*, uint8_t *index*) [inline], [static]

This function sets the upper buffer index upper limitation for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

<i>index</i>	Setting value of upper limitation for buffer index.
--------------	---

7.2.3.21 **static uint8_t DAC_HAL_GetBuffCurrentIndex(uint32_t *baseAddr*) [inline], [static]**

This function gets the current buffer index for the DAC module.

Parameters

<i>baseAddr</i>	The DAC peripheral base address.
-----------------	----------------------------------

Returns

Value of current buffer index.

7.2.3.22 **static void DAC_HAL_SetBuffCurrentIndex(uint32_t *baseAddr*, uint8_t *index*) [inline], [static]**

This function sets the upper buffer index for the DAC module.

Parameters

<i>baseAddr</i>	the DAC peripheral base address.
<i>index</i>	Setting value for buffer index.

7.3 DAC Peripheral Driver

7.3.1 Overview

This section describes the programming interface of the DAC Peripheral driver. The DAC peripheral driver configures the DAC (Digital-to-Analog Converter). It also handles the module initialization and configuration by converting attributes.

7.3.2 DAC Initialization

To initialize the DAC module, call the [DAC_DRV_Init\(\)](#) function and pass the configuration data structure, which can be filled by the [DAC_DRV_StructInitUserConfigNormal\(\)](#) function with the default settings for the converter. After it is initialized, the DAC module can function as a DAC converter.

The DAC module provides advanced features internally with the hardware buffer. To use the advanced features, the API of the [DAC_DRV_EnableBuff\(\)](#) function should be called to initialize the buffer.

7.3.3 DAC Model building

The DAC module provides advanced features with the hardware DAC buffer.

When the DAC is enabled and the buffer is not enabled, the DAC module always converts the data in DAT0, the first item in the buffer, to analog output voltage. If the buffer is enabled, the DAC converts the items in the data buffer to analog output voltage according to the buffer configuration. The data buffer read pointer advances to the next word whenever any hardware or software trigger event occurs. The data buffer can be configured to operate in Normal mode, Swing mode, One-Time Scan mode or FIFO mode:

- Normal mode is the default mode for the buffer. The buffer works as a FIFO buffer. The read pointer increases once triggered. When the read pointer reaches the upper limit, it goes to zero during the next trigger event.
- Swing mode is similar to the FIFO mode. However, when the read pointer reaches the upper limit, it does not go to zero. It will descend by 1 in the next trigger events until reach to zero, and then turns back.
- One-time Scan mode. In One-time Scan mode, the read pointer increases by one once triggered. When it reaches the upper limit, it stops there. If read pointer is reset to the address other than the upper limit, it increases to the upper address and stops. If the software sets the read pointer to the upper limit, the read pointer does not advance in this mode. When the buffer operation is switched from one mode to another, the read pointer does not change.
- FIFO mode. In FIFO mode, the buffer works like a ring FIFO. Some configurations are different from other mode for buffer. The buffer size in used would be set as the max value of it by hardware automatically. The previous upper index will be write pointer of the ring FIFO, while the previous read pointer points to the tail of the ring FIFO. When user pushes the data into FIFO, the write pointer increase. When user trigger the buffer to pop the data, the read pointer increases.

7.3.4 DAC Call diagram

Four kinds of typical use cases are designed for the DAC module:

- Normal converter mode. Normal converter mode is working without the buffer and the trigger. It is the simplest way to use the DAC module.
- Buffer Normal mode. Buffer Normal mode enables the internal buffer and sets it as normal ring buffer mode.
- Buffer Swing mode. Buffer Swing mode enables the internal buffer and set it as Swing mode.
- Buffer One-time Scan mode. One-time Scan buffer mode enables the internal buffer and sets it as One-time Scan mode.
- Buffer FIFO mode. Buffer FIFO works enable the internal buffer and set it as FIFO mode.

These use cases can function on the software trigger or hardware trigger. To use the hardware trigger, enable the hardware trigger setting in the DAC module. Then configure the other module that can generate the trigger, such as the PDB.

These are examples to initialize and configure the DAC driver for typical use cases:

Normal converter mode:

```
// dac_test_normal.c //  
  
#include <stdio.h>  
#include <stdint.h>  
#include <stdbool.h>  
#include "fsl_dac_driver.h"  
#include "fsl_os_abstraction.h"  
  
#define DAC_TEST_BUFF_SIZE (16U)  
  
volatile uint32_t g_dacInstance = 0U;  
  
static uint16_t g_dacBuffDat[DAC_TEST_BUFF_SIZE];  
  
extern void DAC_TEST_FillBuffDat(uint16_t *buffPtr, uint32_t buffLen);  
  
void DAC_TEST_NormalMode(uint32_t instance)  
{  
    dac_user_config_t MyDacUserConfigStruct;  
    uint8_t i;  
  
    g_dacInstance = instance;  
    // Fill values into data buffer. //  
    DAC_TEST_FillBuffDat(g_dacBuffDat, DAC_TEST_BUFF_SIZE);  
  
    // Fill the structure with configuration of software trigger. //  
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);  
  
    // Initialize the DAC Converter. //  
    DAC_DRV_Init(instance, &MyDacUserConfigStruct);  
  
    // Output the DAC value. //  
    for (i = 0U; i < DAC_TEST_BUFF_SIZE; i++)  
    {  
        printf("DAC_DRV_Output: %d\r\n", g_dacInstance[i]);  
        DAC_DRV_Output(instance, g_dacInstance[i]);  
        OSA_TimeDelay(200);  
    }  
  
    // De-initialize the DAC converter. //
```

```

    DAC_DRV_Deinit(instance);
}

```

Buffer Normal mode:

```

// dac_test_buffer_normal.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_dac_driver.h"
#include "fsl_os_abstraction.h"

#define DAC_TEST_BUFF_SIZE (16U)

static dac_state_t MyDacStateStructForBufferNormal;
static uint32_t MyDacIsrCounterBuffStart = 0U;
static uint32_t MyDacIsrCounterBuffUpper = 0U;
static uint32_t MyDacIsrCounterBuffWatermark = 0U;
volatile uint32_t g_dacInstance = 0U;

static uint16_t g_dacBuffDat[DAC_TEST_BUFF_SIZE];

static void DAC_ISR_Buffer(void);
extern void DAC_TEST_FillBuffDat(uint16_t *buffPtr, uint32_t buffLen);

void DAC_TEST_BufferNormalMode(uint32_t instance)
{
    dac_user_config_t MyDacUserConfigStruct;
    dac_buff_config_t MyDacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    g_dacInstance = instance;
    // Fill values into data buffer. //
    DAC_TEST_FillBuffDat(g_dacBuffDat, DAC_TEST_BUFF_SIZE);

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &MyDacUserConfigStruct);

    // Enable the feature of DAC internal buffer. //
    MyDacBuffConfigStruct.bufIndexWatermarkIntEnable = true;
    MyDacBuffConfigStruct.bufIndexStartIntEnable = true;
    MyDacBuffConfigStruct.bufIndexUpperIntEnable = true;
    MyDacBuffConfigStruct.dmaEnable = false;
    MyDacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word
    ;
    MyDacBuffConfigStruct.bufWorkMode = kDacBuffWorkAsNormalMode;
    MyDacBuffConfigStruct.bufUpperIndex = DAC_TEST_BUFF_SIZE - 1U;
    DAC_DRV_EnableBuff(instance, &MyDacBuffConfigStruct, &MyDacStateStructForBufferNormal
    );

    // Fill the buffer with setting data. //
    DAC_DRV_SetBuffValue(instance, 0U, DAC_TEST_BUFF_SIZE, g_dacBuffDat);

    // Register the callback function for DAC buffer event. //
    DAC_DRV_InstallCallback(instance, DAC_ISR_Buffer);

    // Trigger the buffer to output setting value. //
    for (i = 0U; i < DAC_TEST_BUFF_SIZE*2U; i++)
    {
        dacValue = DAC_DRV_SoftTriggerBuff(instance);
        printf("DAC_DRV_SoftTriggerBuff: %d\r\n", dacValue);
    }
}

```

DAC Peripheral Driver

```
    OSA_TimeDelay(200);
}

// Disable the feature of DAC internal buffer. //
DAC_DRV_DisableBuff(instance);

// De-initialize the DAC converter. //
DAC_DRV_Deinit(instance);
}

static void DAC_ISR_Buffer(void)
{
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexStartFlag) )
    {
        MyDacIsrCounterBuffStart++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexWatermarkFlag) )
    {
        MyDacIsrCounterBuffWatermark++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexUpperFlag) )
    {
        MyDacIsrCounterBuffUpper++;
    }
}
```

Buffer Swing mode:

```
// dac_test_buffer_swing.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_dac_driver.h"
#include "fsl_os_abstraction.h"

#define DAC_TEST_BUFF_SIZE (16U)

static dac_state_t MyDacStateStructForBufferSwing;
static uint32_t MyDacIsrCounterBuffStart = 0U;
static uint32_t MyDacIsrCounterBuffUpper = 0U;
static uint32_t MyDacIsrCounterBuffWatermark = 0U;
volatile uint32_t g_dacInstance = 0U;

static uint16_t g_dacBuffDat[DAC_TEST_BUFF_SIZE];

static void DAC_ISR_Buffer(void);
extern void DAC_TEST_FillBuffDat(uint16_t *buffPtr, uint32_t buffLen);

void DAC_TEST_BufferSwingMode(uint32_t instance)
{
    dac_user_config_t MyDacUserConfigStruct;
    dac_buff_config_t MyDacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    g_dacInstance = instance;
    // Fill values into data buffer. //
    DAC_TEST_FillBuffDat(g_dacBuffDat, DAC_TEST_BUFF_SIZE);

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &MyDacUserConfigStruct);
```

```

// Enable the feature of DAC internal buffer. //
MyDacBuffConfigStruct	buffIndexWatermarkIntEnable = true;
MyDacBuffConfigStruct	buffIndexStartIntEnable = true;
MyDacBuffConfigStruct	buffIndexUpperIntEnable = true;
MyDacBuffConfigStruct	dmaEnable = false;
MyDacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word
;
MyDacBuffConfigStruct.bufferMode = kDacBuffWorkAsSwingMode;
MyDacBuffConfigStruct.bufferUpperIndex = DAC_TEST_BUFF_SIZE - 1U;
DAC_DRV_EnableBuff(instance, &MyDacBuffConfigStruct, &MyDacStateStructForBufferSwing)
;

// Fill the buffer with setting data. //
for (i = 0; i < buffLen; i++)
{
    DAC_DRV_SetBuffValue(instance, 0U, DAC_TEST_BUFF_SIZE, g_dacBuffData);
}
// Register the callback function for DAC buffer event. //
DAC_DRV_InstallCallback(instance, DAC_ISR_Buffer);

// Trigger the buffer to output setting value. //
for (i = 0U; i < DAC_TEST_BUFF_SIZE*2U; i++)
{
    dacValue = DAC_DRV_SoftTriggerBuff(instance);
    printf("DAC_DRV_SoftTriggerBuff: %d\r\n", dacValue);
    OSA_TimeDelay(200);
}

// Disable the feature of DAC internal buffer. //
DAC_DRV_DisableBuff(instance);

// De-initialize the DAC converter. //
DAC_DRV_Deinit(instance);
}

static void DAC_ISR_Buffer(void)
{
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexStartFlag) )
    {
        MyDacIsrCounterBuffStart++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexWatermarkFlag) )
    {
        MyDacIsrCounterBuffWatermark++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexUpperFlag) )
    {
        MyDacIsrCounterBuffUpper++;
    }
}

```

Buffer One-time Scan mode:

```

// dac_test_buffer_one_time_scan.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_dac_driver.h"
#include "fsl_os_abstraction.h"

#define DAC_TEST_BUFF_SIZE (16U)

static dac_state_t MyDacStateStructForBufferOneTimeScan;
static uint32_t MyDacIsrCounterBuffStart = 0U;
static uint32_t MyDacIsrCounterBuffUpper = 0U;

```

DAC Peripheral Driver

```
static uint32_t MyDacIsrCounterBuffWatermark = 0U;
volatile uint32_t g_dacInstance = 0U;

static uint16_t g_dacBuffDat[DAC_TEST_BUFF_SIZE];

static void DAC_ISR_Buffer(void);
extern void DAC_TEST_FillBuffDat(uint16_t *buffPtr, uint32_t buffLen);

void DAC_TEST_BufferOneTimeScanMode(uint32_t instance)
{
    dac_user_config_t MyDacUserConfigStruct;
    dac_buff_config_t MyDacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    g_dacInstance = instance;
    // Fill values into data buffer. //
    DAC_TEST_FillBuffDat(g_dacBuffDat, DAC_TEST_BUFF_SIZE);

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &MyDacUserConfigStruct);

    // Enable the feature of DAC internal buffer. //
    MyDacBuffConfigStruct.bufIndexWatermarkIntEnable = true;
    MyDacBuffConfigStruct.bufIndexStartIntEnable = true;
    MyDacBuffConfigStruct.bufIndexUpperIntEnable = true;
    MyDacBuffConfigStruct.dmaEnable = false;
    MyDacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word
    ;
    MyDacBuffConfigStruct.bufWorkMode = kDacBuffWorkAsOneTimeScanMode;
    MyDacBuffConfigStruct.bufUpperIndex = DAC_TEST_BUFF_SIZE - 1U;
    DAC_DRV_EnableBuff(instance, &MyDacBuffConfigStruct, &
        MyDacStateStructForBufferOneTimeScan);

    // Fill the buffer with setting data. //
    DAC_DRV_SetBuffValue(instance, 0U, DAC_TEST_BUFF_SIZE, g_dacBuffDat);

    // Register the callback function for DAC buffer event. //
    DAC_DRV_InstallCallback(instance, DAC_ISR_Buffer);

    // Trigger the buffer to output setting value. //
    for (i = 0U; i < DAC_TEST_BUFF_SIZE*2U; i++)
    {
        dacValue = DAC_DRV_SoftTriggerBuff(instance);
        printf("DAC_DRV_SoftTriggerBuff: %d\r\n", dacValue);
        OSA_TimeDelay(200);
    }

    // Disable the feature of DAC internal buffer. //
    DAC_DRV_DisableBuff(instance);

    // De-initialize the DAC converter. //
    DAC_DRV_Deinit(instance);
}

static void DAC_ISR_Buffer(void)
{
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexStartFlag) )
    {
        MyDacIsrCounterBuffStart++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexWatermarkFlag) )
    {
        MyDacIsrCounterBuffWatermark++;
    }
}
```

```

if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexUpperFlag) )
{
    MyDacIsrCounterBuffUpper++;
}
}

```

Buffer FIFO mode:

```

// dac_test_buffer_fifo.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_dac_driver.h"
#include "fsl_os_abstraction.h"

#define DAC_TEST_BUFF_SIZE (16U)

static dac_state_t MyDacStateStructForBufferFIFO;
static uint32_t MyDacIsrCounterBuffStart = 0U;
static uint32_t MyDacIsrCounterBuffUpper = 0U;
static uint32_t MyDacIsrCounterBuffWatermark = 0U;
volatile uint32_t g_dacInstance = 0U;
volatile uint32_t g_dacBuffIndex = 0U;

static uint16_t g_dacBuffDat[DAC_TEST_BUFF_SIZE];

static void DAC_ISR_Buffer(void);
extern void DAC_TEST_FillBuffDat(uint16_t *buffPtr, uint32_t buffLen);

void DAC_TEST_BufferFIFOMode(uint32_t instance)
{
    dac_user_config_t MyDacUserConfigStruct;
    dac_buff_config_t MyDacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    g_dacInstance = instance;
    // Fill values into data buffer. //
    DAC_TEST_FillBuffDat(g_dacBuffDat, DAC_TEST_BUFF_SIZE);

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &MyDacUserConfigStruct);

    // Configure the feature of DAC internal buffer. //
    MyDacBuffConfigStruct.buffIndexWatermarkIntEnable = true;
    MyDacBuffConfigStruct.buffIndexStartIntEnable = true;
    MyDacBuffConfigStruct.buffIndexUpperIntEnable = true;
    MyDacBuffConfigStruct.dmaEnable = false;
    MyDacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word
    ;
    MyDacBuffConfigStruct.buffWorkMode = kDacBuffWorkAsFIFOMode;
    MyDacBuffConfigStruct.buffUpperIndex = 0U; // Write buffer pointer. //

    // Note:
    // Since the FIFO buffer's interrupt flags can not be cleared by writing
    // the status register, application should fill the initial user-defined
    // callback function before enable the buffer as FIFO mode. After that,
    // user can use DAC_DRV_InstallCallback() to update callback function.
    //
    MyDacStateStructForBufferFIFO.userCallbackFunc = DAC_ISR_Buffer;

    // Enable the DAC buffer. //
}

```

DAC Peripheral Driver

```
DAC_DRV_EnableBuff(instance, &MyDacBuffConfigStruct, &MyDacStateStructForBufferFIFO);

// Trigger the buffer to output setting value. //
for (i = 0U; i < DAC_TEST_BUFSIZE*2U; i++)
{
    dacValue = DAC_DRV_SoftTriggerBuff(instance);
    printf("DAC_DRV_SoftTriggerBuff: %d\r\n", dacValue);
    OSA_TimeDelay(200);
}

// Disable the feature of DAC internal buffer. //
DAC_DRV_DisableBuff(instance);

// De-initialize the DAC converter. //
DAC_DRV_Deinit(instance);
}

static void DAC_ISR_Buffer(void)
{
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexStartFlag) )
    {
        MyDacIsrCounterBuffStart++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexWatermarkFlag) )
    {
        // When the FIFO's data count is less then the watermark, it would be filled here automatically.
        //
        if (g_dacBuffIndex >= DAC_TEST_BUFSIZE)
        {
            g_dacBuffIndex = 0U;
        }
        DAC_DRV_SetBuffValue(g_dacInstance, 0U, 1U, &g_dacBuffDat[g_dacBuffIndex++]);
        MyDacIsrCounterBuffWatermark++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexUpperFlag) )
    {
        MyDacIsrCounterBuffUpper++;
    }
}
```

Data Structures

- struct `dac_buff_config_t`
Defines the configuration buffer structure inside the DAC module. [More...](#)
- struct `dac_user_config_t`
Defines the converted configuration structure. [More...](#)

Enumerations

- enum `dac_flag_t` {
 kDacBuffIndexStartFlag = 1U,
 kDacBuffIndexUpperFlag = 2U }
Defines the type of event flags.

Functions

- **dac_status_t DAC_DRV_StructInitUserConfigNormal (dac_user_config_t *userConfigPtr)**
Populates the initial user configuration for the DAC module without interrupt and buffer features.
- **dac_status_t DAC_DRV_Init (uint32_t instance, dac_user_config_t *userConfigPtr)**
Initializes the converter.
- **void DAC_DRV_Deinit (uint32_t instance)**
De-initializes the DAC module converter.
- **void DAC_DRV_Output (uint32_t instance, uint16_t value)**
Drives the converter to output the DAC value.
- **dac_status_t DAC_DRV_EnableBuff (uint32_t instance, dac_buff_config_t *buffConfigPtr)**
Configures the internal buffer.
- **void DAC_DRV_DisableBuff (uint32_t instance)**
Disables the internal buffer.
- **dac_status_t DAC_DRV_SetBuffValue (uint32_t instance, uint8_t start, uint8_t offset, uint16_t arr[])**
Sets values into the DAC internal buffer.
- **uint16_t DAC_DRV_SoftTriggerBuff (uint32_t instance)**
Triggers the buffer by software and returns the current value.
- **uint8_t DAC_DRV_GetBufferIndex (uint32_t instance)**
Gets the DAC buffer current index.
- **void DAC_DRV_ClearFlag (uint32_t instance, dac_flag_t flag)**
Clears the flag for an indicated event causing an interrupt.
- **bool DAC_DRV_GetFlag (uint32_t instance, dac_flag_t flag)**
Gets the flag for an indicated event causing an interrupt.

Variables

- **const uint32_t g_dacBaseAddr []**
Table of base addresses for DAC instances.
- **const IRQn_Type g_dacIrqId [HW_DAC_INSTANCE_COUNT]**
Table to save DAC IRQ enumeration numbers defined in the CMSIS header file.

7.3.5 Data Structure Documentation

7.3.5.1 struct dac_buff_config_t

This structure holds the configuration for the internal buffer inside the DAC module. The DAC buffer is an advanced feature, which helps improve the performance of an application.

Data Fields

- **bool buffIndexStartIntEnable**
Switcher to enable interrupt when buffer index hits the start (0).
- **bool buffIndexUpperIntEnable**
Switcher to enable interrupt when buffer index hits the upper.

DAC Peripheral Driver

- **bool dmaEnable**
Switcher to enable DMA request by original interrupts.
- **dac_buff_work_mode_t buffWorkMode**
Selection of buffer's work mode.

7.3.5.1.0.11 Field Documentation

7.3.5.1.0.11.1 **bool dac_buff_config_t::buffIndexStartIntEnable**

7.3.5.1.0.11.2 **bool dac_buff_config_t::buffIndexUpperIntEnable**

7.3.5.1.0.11.3 **bool dac_buff_config_t::dmaEnable**

7.3.5.1.0.11.4 **dac_buff_work_mode_t dac_buff_config_t::buffWorkMode**

See "dac_buff_work_mode_t".

7.3.5.2 struct dac_user_config_t

This structure holds the configuration for DAC module basic converter. The DAC converter is the core part of the DAC module. When initialized, the DAC module can act as a simple DAC converter.

Data Fields

- **dac_ref_volt_src_mode_t refVoltSrcMode**
Selection of reference voltage source for DAC module.
- **dac_trigger_mode_t triggerMode**
Selection of hardware mode or software mode.
- **bool lowPowerEnable**
Switcher to enable working in low power mode.

7.3.5.2.0.12 Field Documentation

7.3.5.2.0.12.1 **dac_ref_volt_src_mode_t dac_user_config_t::refVoltSrcMode**

7.3.5.2.0.12.2 **dac_trigger_mode_t dac_user_config_t::triggerMode**

7.3.5.2.0.12.3 **bool dac_user_config_t::lowPowerEnable**

7.3.6 Enumeration Type Documentation

7.3.6.1 enum dac_flag_t

Enumerator

kDacBuffIndexStartFlag Event for the buffer index hit the start (0).

kDacBuffIndexUpperFlag Event for the buffer index hit the upper.

7.3.7 Function Documentation

7.3.7.1 `dac_status_t DAC_DRV_StructInitUserConfigNormal (dac_user_config_t * userConfigPtr)`

This function populates the initial user configuration without interrupt and buffer features. Calling the initialization function with the populated parameter configures the DAC module to operate as a simple converter. The settings are:

- .refVoltSrcMode = kDacRefVoltSrcOfVref2; // Vdda
- .triggerMode = kDacTriggerBySoftware;
- .lowPowerEnable = false;

Parameters

<i>userConfigPtr</i>	Pointer to the user configuration structure. See the "dac_user_config_t".
----------------------	---

Returns

Execution status.

7.3.7.2 `dac_status_t DAC_DRV_Init (uint32_t instance, dac_user_config_t * userConfigPtr)`

This function initializes the converter. It configures the DAC converter itself but does not include the advanced features, such as interrupt and internal buffer. This API should be called before any operations in the DAC module. After it is initialized, the DAC module can function as a simple DAC converter.

Parameters

<i>instance</i>	DAC instance ID.
<i>userConfigPtr</i>	Pointer to the initialization structure. See the "dac_user_config_t".

Returns

Execution status.

7.3.7.3 `void DAC_DRV_Deinit (uint32_t instance)`

This function de-initializes the converter. It disables the DAC module and shuts down the clock to reduce the power consumption.

DAC Peripheral Driver

Parameters

<i>instance</i>	DAC instance ID.
-----------------	------------------

7.3.7.4 void DAC_DRV_Output (uint32_t *instance*, uint16_t *value*)

This function drives the converter to output the DAC value. It forces the buffer index to be the first one and load the setting value to this item. Then, the converter outputs the voltage indicated by the indicated value immediately.

Parameters

<i>instance</i>	DAC instance ID.
<i>value</i>	Setting value for DAC.

7.3.7.5 dac_status_t DAC_DRV_EnableBuff (uint32_t *instance*, dac_buff_config_t * *buffConfigPtr*)

This function configures the feature of the internal buffer for the DAC module. By default, the buffer feature is disabled. Calling this API enables the buffer and configures it.

Parameters

<i>instance</i>	DAC instance ID.
<i>buffConfigPtr</i>	Pointer to the configuration structure. See the "dac_buff_config_t".

Returns

Execution status.

7.3.7.6 void DAC_DRV_DisableBuff (uint32_t *instance*)

This function disables the internal buffer feature. Calling this API disables the internal buffer feature and resets the DAC module as a simple DAC converter.

Parameters

<i>instance</i>	DAC instance ID.
-----------------	------------------

7.3.7.7 **dac_status_t DAC_DRV_SetBuffValue (uint32_t *instance*, uint8_t *start*, uint8_t *offset*, uint16_t *arr*[])**

This function sets values into the DAC internal buffer. Note that the buffer size is defined by the "FSL_FEATURE_DAC_BUFFER_SIZE" macro and the available value is 12 bit.

Parameters

<i>instance</i>	DAC instance ID.
<i>start</i>	Start index of setting values.
<i>offset</i>	Length of setting values' array.
<i>arr</i>	Setting values' array.

Returns

Execution status.

7.3.7.8 **uint16_t DAC_DRV_SoftTriggerBuff (uint32_t *instance*)**

This function triggers the buffer by software and returns the current value. After it is triggered, the buffer index updates according to work mode. Then, the value kept inside the pointed item is immediately output.

Parameters

<i>instance</i>	DAC instance ID.
-----------------	------------------

Returns

Current output value.

7.3.7.9 **uint8_t DAC_DRV_GetBufferIndex (uint32_t *instance*)**

This function gets the DAC buffer current index.

DAC Peripheral Driver

Parameters

<i>instance</i>	DAC instance ID.
-----------------	------------------

Returns

Current index of DAC buffer.

7.3.7.10 void DAC_DRV_ClearFlag (uint32_t *instance*, dac_flag_t *flag*)

This function clears the flag for an indicated event causing an interrupt.

Parameters

<i>instance</i>	DAC instance ID.
<i>flag</i>	Indicated flag. See "dac_flag_t".

7.3.7.11 bool DAC_DRV_GetFlag (uint32_t *instance*, dac_flag_t *flag*)

This function gets the flag for an indicated event causing an interrupt. If the event occurs, the return value is asserted.

Parameters

<i>instance</i>	DAC instance ID.
<i>flag</i>	Indicated flag. See "dac_flag_t".

Returns

Assertion of indicated event.

7.3.8 Variable Documentation

7.3.8.1 const uint32_t g_dacBaseAddr[]

7.3.8.2 const IRQn_Type g_dacIrqId[HW_DAC_INSTANCE_COUNT]

Chapter 8

Direct Memory Access (DMA)

8.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Direct Memory Access block of Kinetis devices.

Modules

- DMA HAL driver
- DMA driver
- DMA request
- DMAMUX HAL driver

DMA HAL driver

8.2 DMA HAL driver

8.2.1 Overview

This section describes the programming interface of the DMA HAL driver.

Data Structures

- struct [dma_channel_link_config_t](#)
Data structure for data structure configuration. [More...](#)
- union [dma_error_status_t](#)
Data structure to get status of the DMA channel status. [More...](#)

Enumerations

- enum [dma_status_t](#) { ,
 kStatus_DMA_InvalidArgument = 1U,
 kStatus_DMA_Fail = 2U }
DMA status.
- enum [dma_transfer_size_t](#) {
 kDmaTransferSize32bits = 0x0U,
 kDmaTransferSize8bits = 0x1U,
 kDmaTransferSize16bits = 0x2U }
DMA transfer size type.
- enum [dma_modulo_t](#)
Configuration type for the DMA modulo.
- enum [dma_channel_link_type_t](#) {
 kDmaChannelLinkDisable = 0x0U,
 kDmaChannelLinkChan1AndChan2 = 0x1U,
 kDmaChannelLinkChan1 = 0x2U,
 kDmaChannelLinkChan1AfterBCR0 = 0x3 }
DMA channel link type.
- enum [dma_transfer_type_t](#) {
 kDmaPeripheralToMemory,
 kDmaMemoryToPeripheral,
 kDmaMemoryToMemory,
 kDmaPeripheralToPeripheral }
Type for DMA transfer.

DMA HAL channel configuration

- void [DMA_HAL_Init](#) (uint32_t baseAddr, uint32_t channel)
Sets all registers of the channel to 0.
- void [DMA_HAL_ConfigTransfer](#) (uint32_t baseAddr, uint32_t channel, [dma_transfer_size_t](#) size, [dma_transfer_type_t](#) type, uint32_t sourceAddr, uint32_t destAddr, uint32_t length)

- Sets all registers of the channel to 0.
- static void **DMA_HAL_SetSourceAddr** (uint32_t baseAddr, uint32_t channel, uint32_t address)
Configures the source address.
- static void **DMA_HAL_SetDestAddr** (uint32_t baseAddr, uint32_t channel, uint32_t address)
Configures the source address.
- static void **DMA_HAL_SetTransferCount** (uint32_t baseAddr, uint32_t channel, uint32_t count)
Configures the bytes to be transferred.
- static uint32_t **DMA_HAL_GetUnfinishedByte** (uint32_t baseAddr, uint32_t channel)
Gets the left bytes not to be transferred.
- static void **DMA_HAL_SetIntCmd** (uint32_t baseAddr, uint8_t channel, bool enable)
Enables the interrupt for the DMA channel after the work is done.
- static void **DMA_HAL_SetCycleStealCmd** (uint32_t baseAddr, uint8_t channel, bool enable)
Configures the DMA transfer mode to cycle steal or continuous modes.
- static void **DMA_HAL_SetAutoAlignCmd** (uint32_t baseAddr, uint8_t channel, bool enable)
Configures the auto-align feature.
- static void **DMA_HAL_SetAsyncDmaRequestCmd** (uint32_t baseAddr, uint8_t channel, bool enable)
Configures the a-sync DMA request feature.
- static void **DMA_HAL_SetSourceIncrementCmd** (uint32_t baseAddr, uint32_t channel, bool enable)
Enables/Disables the source increment.
- static void **DMA_HAL_SetDestIncrementCmd** (uint32_t baseAddr, uint32_t channel, bool enable)
Enables/Disables destination increment.
- static void **DMA_HAL_SetSourceTransferSize** (uint32_t baseAddr, uint32_t channel, **dma_transfer_size_t** transfersize)
Configures the source transfer size.
- static void **DMA_HAL_SetDestTransferSize** (uint32_t baseAddr, uint32_t channel, **dma_transfer_size_t** transfersize)
Configures the destination transfer size.
- static void **DMA_HAL_SetTriggerStartCmd** (uint32_t baseAddr, uint32_t channel, bool enable)
Triggers the start.
- static void **DMA_HAL_SetSourceModulo** (uint32_t baseAddr, uint32_t channel, **dma_modulo_t** modulo)
Configures the modulo for the source address.
- static void **DMA_HAL_SetDestModulo** (uint32_t baseAddr, uint32_t channel, **dma_modulo_t** modulo)
Configures the modulo for the destination address.
- static void **DMA_HAL_SetDmaRequestCmd** (uint32_t baseAddr, uint32_t channel, bool enable)
Enables/Disables the DMA request.
- static void **DMA_HAL_SetDisableRequestAfterDoneCmd** (uint32_t baseAddr, uint32_t channel, bool enable)
Configures the DMA request state after the work is done.
- void **DMA_HAL_SetChanLink** (uint32_t baseAddr, uint8_t channel, **dma_channel_link_config_t** *mode)
Configures the channel link feature.
- static void **DMA_HAL_ClearStatus** (uint32_t baseAddr, uint8_t channel)
Clears the status of the DMA channel.
- static **dma_error_status_t** **DMA_HAL_GetStatus** (uint32_t baseAddr, uint8_t channel)
Gets the DMA controller channel status.

8.2.2 Data Structure Documentation

8.2.2.1 struct dma_channel_link_config_t

Data Fields

- **dma_channel_link_type_t linkType**
Channel link type.
- **uint32_t channel1**
Channel 1 configuration.
- **uint32_t channel2**
Channel 2 configuration.

8.2.2.2 union dma_error_status_t

8.2.2.2.0.13 Field Documentation

8.2.2.2.0.13.1 uint32_t dma_error_status_t::dmaTransDone

8.2.2.2.0.13.2 uint32_t dma_error_status_t::dmaBusy

8.2.2.2.0.13.3 uint32_t dma_error_status_t::dmaPendingRequest

8.2.3 Enumeration Type Documentation

8.2.3.1 enum dma_status_t

Enumerator

kStatus_DMA_InvalidArgument Parameter is not available for the current configuration.

kStatus_DMA_Fail Function operation failed.

8.2.3.2 enum dma_transfer_size_t

Enumerator

kDmaTransferSize32bits 32 bits are transferred for every read/write

kDmaTransferSize8bits 8 bits are transferred for every read/write

kDmaTransferSize16bits 16b its are transferred for every read/write

8.2.3.3 enum dma_channel_link_type_t

Enumerator

kDmaChannelLinkDisable No channel link.

kDmaChannelLinkChan1AndChan2 Perform a link to channel 1 after each cycle-steal transfer followed by a link and to channel 2 after the BCR decrements to zeros.

kDmaChannelLinkChan1 Perform a link to channel 1 after each cycle-steal transfer.

kDmaChannelLinkChan1AfterBCR0 Perform a link to channel1 after the BCR decrements to zero.

8.2.3.4 enum dma_transfer_type_t

Enumerator

kDmaPeripheralToMemory Transfer from the peripheral to memory.

kDmaMemoryToPeripheral Transfer from the memory to peripheral.

kDmaMemoryToMemory Transfer from the memory to memory.

kDmaPeripheralToPeripheral Transfer from the peripheral to peripheral.

8.2.4 Function Documentation

8.2.4.1 void DMA_HAL_Init (uint32_t *baseAddr*, uint32_t *channel*)

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.

8.2.4.2 void DMA_HAL_ConfigTransfer (uint32_t *baseAddr*, uint32_t *channel*, dma_transfer_size_t *size*, dma_transfer_type_t *type*, uint32_t *sourceAddr*, uint32_t *destAddr*, uint32_t *length*)

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>size</i>	Size to be transferred on each DMA write/read. Source/Dest share the same write/read size.

DMA HAL driver

<i>type</i>	Transfer type.
<i>sourceAddr</i>	Source address.
<i>destAddr</i>	Destination address.
<i>length</i>	Bytes to be transferred.

8.2.4.3 static void DMA_HAL_SetSourceAddr (*uint32_t baseAddr, uint32_t channel, uint32_t address*) [inline], [static]

Each SAR contains the byte address used by the DMA to read data. The SARn is typically aligned on a 0-modulo-size boundary—that is on the natural alignment of the source data. Bits 31-20 of this register must be written with one of the only four allowed values. Each of these allowed values corresponds to a valid region of the devices’ memory map. The allowed values are: 0x000x_xxxx 0x1FFx_xxxx 0x200x_xxxx 0x400x_xxxx After they are written with one of the allowed values, bits 31-20 read back as the written value. After they are written with any other value, bits 31-20 read back as an indeterminate value.

This function enables the request for a specified channel.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>address</i>	memory address pointing to the source address.

8.2.4.4 static void DMA_HAL_SetDestAddr (*uint32_t baseAddr, uint32_t channel, uint32_t address*) [inline], [static]

Each DAR contains the byte address used by the DMA to read data. The DARn is typically aligned on a 0-modulo-size boundary—that is on the natural alignment of the source data. Bits 31-20 of this register must be written with one of the only four allowed values. Each of these allowed values corresponds to a valid region of the devices’ memory map. The allowed values are: 0x000x_xxxx 0x1FFx_xxxx 0x200x_xxxx 0x400x_xxxx After they are written with one of the allowed values, bits 31-20 read back as the written value. After they are written with any other value, bits 31-20 read back as an indeterminate value.

This function enables the request for specified channel.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>address</i>	Destination address.

8.2.4.5 static void DMA_HAL_SetTransferCount(uint32_t *baseAddr*, uint32_t *channel*, uint32_t *count*) [inline], [static]

Transfer bytes must be written with a value equal to or less than 0F_FFFFh. After being written with a value in this range, bits 23-20 of the BCR read back as 1110b. A write to the BCR with a value greater than 0F_FFFFh causes a configuration error when the channel starts to execute. After they are written with a value in this range, bits 23-20 of BCR read back as 1111b.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>count</i>	bytes to be transferred.

8.2.4.6 static uint32_t DMA_HAL_GetUnfinishedByte(uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.

Returns

unfinished bytes.

8.2.4.7 static void DMA_HAL_SetIntCmd(uint32_t *baseAddr*, uint8_t *channel*, bool *enable*) [inline], [static]

This function enables the request for specified channel.

DMA HAL driver

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	True means enable interrupt, false means disable.

8.2.4.8 static void DMA_HAL_SetCycleStealCmd (*uint32_t baseAddr, uint8_t channel, bool enable*) [inline], [static]

If continuous mode is enabled, DMA continuously makes write/read transfers until BCR decrement to 0. If continuous mode is disabled, DMA write/read is only triggered on every request. s

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	1 means cycle-steal mode, 0 means continuous mode.

8.2.4.9 static void DMA_HAL_SetAutoAlignCmd (*uint32_t baseAddr, uint8_t channel, bool enable*) [inline], [static]

If auto-align is enabled, the appropriate address register increments, regardless of whether it is a source increment or a destination increment.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	0 means disable auto-align. 1 means enable auto-align.

8.2.4.10 static void DMA_HAL_SetAsyncDmaRequestCmd (*uint32_t baseAddr, uint8_t channel, bool enable*) [inline], [static]

Enables/Disables the a-synchronization mode in a STOP mode for each DMA channel.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	0 means disable DMA request a-sync. 1 means enable DMA request -.

8.2.4.11 static void DMA_HAL_SetSourceIncrementCmd (*uint32_t baseAddr, uint32_t channel, bool enable*) [inline], [static]

Controls whether the source address increments after each successful transfer. If enabled, the SAR increments by 1,2,4 as determined by the transfer size.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	Enabled/Disable increment.

8.2.4.12 static void DMA_HAL_SetDestIncrementCmd (*uint32_t baseAddr, uint32_t channel, bool enable*) [inline], [static]

Controls whether the destination address increments after each successful transfer. If enabled, the DAR increments by 1,2,4 as determined by the transfer size.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	Enabled/Disable increment.

8.2.4.13 static void DMA_HAL_SetSourceTransferSize (*uint32_t baseAddr, uint32_t channel, dma_transfer_size_t transfersize*) [inline], [static]

Parameters

<i>baseAddr</i>	DMA baseAddr.
-----------------	---------------

DMA HAL driver

<i>channel</i>	DMA channel.
<i>transfersize</i>	enum type for transfer size.

8.2.4.14 static void DMA_HAL_SetDestTransferSize (uint32_t *baseAddr*, uint32_t *channel*, dma_transfer_size_t *transfersize*) [inline], [static]

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>transfersize</i>	enum type for transfer size.

8.2.4.15 static void DMA_HAL_SetTriggerStartCmd (uint32_t *baseAddr*, uint32_t *channel*, bool *enable*) [inline], [static]

When the DMA begins the transfer, the START bit is cleared automatically after one module clock and always reads as logic 0.

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	Enable/disable trigger start.

8.2.4.16 static void DMA_HAL_SetSourceModulo (uint32_t *baseAddr*, uint32_t *channel*, dma_modulo_t *modulo*) [inline], [static]

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>modulo</i>	enum data type for source modulo.

8.2.4.17 static void DMA_HAL_SetDestModulo (uint32_t *baseAddr*, uint32_t *channel*, dma_modulo_t *modulo*) [inline], [static]

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>modulo</i>	enum data type for dest modulo.

8.2.4.18 static void DMA_HAL_SetDmaRequestCmd (*uint32_t baseAddr, uint32_t channel, bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	DMA baseAddr.
<i>channel</i>	DMA channel.
<i>enable</i>	Enable/disable dma request.

8.2.4.19 static void DMA_HAL_SetDisableRequestAfterDoneCmd (*uint32_t baseAddr, uint32_t channel, bool enable*) [inline], [static]

Disables/Enables the DMA request after a DMA DONE is generated. If it works in the loop mode, this bit should not be set.

Parameters

<i>baseAddr</i>	DMA base address.
<i>channel</i>	DMA channel.
<i>enable</i>	0 means DMA request would not be disabled after work done. 1 means disable.

8.2.4.20 void DMA_HAL_SetChanLink (*uint32_t baseAddr, uint8_t channel, dma_channel_link_config_t * mode*)

Parameters

<i>baseAddr</i>	DMA base address.
-----------------	-------------------

DMA HAL driver

<i>channel</i>	DMA channel.
<i>mode</i>	Mode of channel link in DMA.

8.2.4.21 static void DMA_HAL_ClearStatus (uint32_t *baseAddr*, uint8_t *channel*) [inline], [static]

This function clears the status for a specified DMA channel. The error status and done status are cleared.

Parameters

<i>baseAddr</i>	DMA base address.
<i>channel</i>	DMA channel.

8.2.4.22 static dma_error_status_t DMA_HAL_GetStatus (uint32_t *baseAddr*, uint8_t *channel*) [inline], [static]

Gets the status of the DMA channel. The user can get the error status, as to whether the descriptor is finished or there are bytes left.

Parameters

<i>baseAddr</i>	DMA base address.
<i>channel</i>	DMA channel.

Returns

Status of the DMA channel.

8.3 DMAMUX HAL driver

8.3.1 Overview

This section describes the programming interface of the DMAMUX HAL module.

Enumerations

- enum `dmamux_dma_request_source` { `kDmamuxDmaRequestSource` = 64U }
- A constant for the length of the DMA hardware source.*

DMAMUX HAL function

- void `DMAMUX_HAL_Init` (uint32_t baseAddr)
Initializes the DMAMUX module to the reset state.
- static void `DMAMUX_HAL_SetChannelCmd` (uint32_t baseAddr, uint32_t channel, bool enable)
Enables/Disables the DMAMUX channel.
- static void `DMAMUX_HAL_SetTriggerSource` (uint32_t baseAddr, uint32_t channel, uint8_t source)
Configures the DMA request for the DMAMUX channel.

8.3.2 Enumeration Type Documentation

8.3.2.1 enum dmamux_dma_request_source

This structure is used inside the DMA driver.

Enumerator

kDmamuxDmaRequestSource Maximum number of the DMA requests allowed for the DMA mux.

8.3.3 Function Documentation

8.3.3.1 void `DMAMUX_HAL_Init` (`uint32_t baseAddr`)

Initializes the DMAMUX module to the reset state.

Parameters

DMAMUX HAL driver

<i>baseAddr</i>	Register base address for DMAMUX module.
-----------------	--

8.3.3.2 static void DMAMUX_HAL_SetChannelCmd (*uint32_t baseAddr, uint32_t channel, bool enable*) [inline], [static]

Enables the hardware request. If enabled, the hardware request is sent to the corresponding DMA channel.

Parameters

<i>baseAddr</i>	Register base address for DMAMUX module.
<i>channel</i>	DMAMUX channel number.
<i>enable</i>	Enables (true) or Disables (false) DMAMUX channel.

8.3.3.3 static void DMAMUX_HAL_SetTriggerSource (*uint32_t baseAddr, uint32_t channel, uint8_t source*) [inline], [static]

Sets the trigger source for the DMA channel. The trigger source is in the file fsl_dma_request.h.

Parameters

<i>baseAddr</i>	Register base address for DMAMUX module.
<i>channel</i>	DMAMUX channel number.
<i>source</i>	DMA request source.

8.4 DMA driver

8.4.1 Overview

This chapter describes the programming interface of the DMA Peripheral driver. The DMA driver requests, configures, and uses the DMA hardware. It supports module initialization and DMA channel configuration.

Data Structures

- struct [dma_channel_t](#)
Data structure for the DMA channel management. [More...](#)
- struct [dma_state_t](#)
Data structure for the DMA controller management. [More...](#)

Typedefs

- [typedef void\(* dma_callback_t \)](#)(void *parameter, [dma_channel_status_t](#) status)
A definition for the DMA channel callback function.

Enumerations

- enum [dma_channel_status_t](#) {
 kDmaIdle,
 kDmaNormal,
 kDmaError }
Channel status for the DMA channel.
- enum [dma_channel_type_t](#) {
 kDmaInvalidChannel = 0xFFU,
 kDmaAnyChannel = 0xFEU }
Type for the DMA channel, which is used for the DMA channel allocation.

Variables

- const uint32_t [g_dmaRegBaseAddr](#) [HW_DMA_INSTANCE_COUNT]
Array for eDMA module register base address.
- const uint32_t [g_dmamuxRegBaseAddr](#) [HW_DMAMUX_INSTANCE_COUNT]
Array for DMAMUX module register base address.
- const IRQn_Type [g_dmaIrqId](#) [HW_DMA_INSTANCE_COUNT][FSL_FEATURE_DMA_DMAMUX_CHANNELS]
Two-dimensional array for EDMA channel interrupt vector number.

DMA driver

DMA Driver

- **dma_status_t DMA_DRV_Init (dma_state_t *state)**
Initializes the DMA.
- **dma_status_t DMA_DRV_Deinit (void)**
De-initializes the DMA.
- **dma_status_t DMA_DRV_RegisterCallback (dma_channel_t *chn, dma_callback_t callback, void *para)**
Registers the callback function and a parameter.
- **uint32_t DMA_DRV_GetUnfinishedBytes (dma_channel_t *chn)**
Gets the number of unfinished bytes.
- **dma_status_t DMA_DRV_ClaimChannel (uint32_t channel, dma_request_source_t source, dma_channel_t *chn)**
Claims a DMA channel.
- **uint32_t DMA_DRV_RequestChannel (uint32_t channel, dma_request_source_t source, dma_channel_t *chn)**
Requests a DMA channel.
- **dma_status_t DMA_DRV_FreeChannel (dma_channel_t *chn)**
Frees DMA channel hardware and software resource.
- **dma_status_t DMA_DRV_StartChannel (dma_channel_t *chn)**
Starts a DMA channel.
- **dma_status_t DMA_DRV_StopChannel (dma_channel_t *chn)**
Stops a DMA channel.
- **dma_status_t DMA_DRV_ConfigTransfer (dma_channel_t *chn, dma_transfer_type_t type, uint32_t size, uint32_t sourceAddr, uint32_t destAddr, uint32_t length)**
Configures a transfer for the DMA.
- **dma_status_t DMA_DRV_ConfigChanLink (dma_channel_t *chn, dma_channel_link_config_t *link_config)**
Configures the channel link feature.
- **void DMA_DRV_IRQHandler (uint32_t channel)**
DMA IRQ handler for both an interrupt and an error.

8.4.2 Data Structure Documentation

8.4.2.1 struct dma_channel_t

Data Fields

- **uint8_t channel**
Channel number.
- **uint8_t dmamuxModule**
Dmamux module index.
- **uint8_t dmamuxChannel**
Dmamux module channel.
- **dma_callback_t callback**
Callback function for this channel.
- **void * parameter**
Parameter for the callback function.

- volatile `dma_channel_status_t` `status`
Channel status.

8.4.2.2 `struct dma_state_t`

8.4.3 Typedef Documentation

8.4.3.1 `typedef void(* dma_callback_t)(void *parameter, dma_channel_status_t status)`

A prototype for the callback function registered into the DMA driver.

8.4.4 Enumeration Type Documentation

8.4.4.1 `enum dma_channel_status_t`

A structure describing the status of the DMA channel. The user can get the status from the channel callback function.

Enumerator

kDmaIdle DMA channel is idle.

kDmaNormal DMA channel is occupied.

kDmaError Error occurs in the DMA channel.

8.4.4.2 `enum dma_channel_type_t`

Enumerator

kDmaInvalidChannel Macros indicating the failure of the channel request.

kDmaAnyChannel Macros used when requesting a channel. `kEdmaAnyChannel` means a request of dynamic channel allocation.

8.4.5 Function Documentation

8.4.5.1 `dma_status_t DMA_DRV_Init (dma_state_t * state)`

Parameters

DMA driver

<i>state</i>	DMA state structure used for the DMA internal logic.
--------------	--

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.2 **dma_status_t DMA_DRV_Deinit (void)**

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.3 **dma_status_t DMA_DRV_RegisterCallback (dma_channel_t * *chn*, dma_callback_t *callback*, void * *para*)**

The user registers the callback function and a parameter for a specified DMA channel. When the channel interrupt or a channel error happens, the callback and the parameter are called. The user parameter is also provided to give a channel status.

Parameters

<i>chn</i>	A handler for the DMA channel
<i>callback</i>	Callback function
<i>para</i>	A parameter for callback functions

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.4 **uint32_t DMA_DRV_GetUnfinishedBytes (dma_channel_t * *chn*)**

Gets the bytes that remain to be transferred.

Parameters

<i>chn</i>	A handler for the DMA channel
------------	-------------------------------

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.5 **dma_status_t DMA_DRV_ClaimChannel (uint32_t *channel*, dma_request_source_t *source*, dma_channel_t * *chn*)**

Parameters

<i>channel</i>	Channel index which needs to claim.
<i>source</i>	DMA request source.
<i>chn</i>	A handler for the DMA channel

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.6 **uint32_t DMA_DRV_RequestChannel (uint32_t *channel*, dma_request_source_t *source*, dma_channel_t * *chn*)**

This function provides two ways to allocate a DMA channel: static and dynamic allocation. To allocate a channel dynamically, set the channel parameter with the value of kDmaAnyChannel. The driver searches all available free channels and assigns the first channel to the user. To allocate the channel statically, set the channel parameter with the value of a specified channel. If the channel is available, the driver assigns the channel. Notes: The user must provide a handler memory for the DMA channel. The driver initializes the handler and configures the handler memory.

Parameters

<i>channel</i>	A DMA channel number. If a channel is assigned with a valid channel number, the DMA driver tries to assign a specified channel. If a channel is assigned with kDmaAnyChannel, the DMA driver searches all available channels and assigns the first channel to the user.
<i>source</i>	A DMA hardware request.
<i>chn</i>	Memory pointing to DMA channel. The user must ensure that the handler memory is valid and that it will not be released or changed by any other code before the channel <code>dma_free_channel()</code> operation.

Returns

If the channel allocation is successful, the return value indicates the requested channel. If not, the driver returns a kDmaInvalidChannel value to indicate that the request operation has failed.

8.4.5.7 **dma_status_t DMA_DRV_FreeChannel (dma_channel_t * *chn*)**

This function frees the relevant software and hardware resources. Both the request and the free operations need to be called as a pair.

DMA driver

Parameters

<i>chn</i>	Memory pointing to DMA channel.
------------	---------------------------------

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.8 dma_status_t DMA_DRV_StartChannel (*dma_channel_t * chn*)

Starts a DMA channel. The driver starts a DMA channel by enabling the DMA request. A software start bit is not used in the DMA Peripheral driver.

Parameters

<i>chn</i>	Memory pointing to the DMA channel.
------------	-------------------------------------

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.9 dma_status_t DMA_DRV_StopChannel (*dma_channel_t * chn*)

Parameters

<i>chn</i>	Memory pointing to the DMA channel.
------------	-------------------------------------

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.10 dma_status_t DMA_DRV_ConfigTransfer (*dma_channel_t * chn,* *dma_transfer_type_t type, uint32_t size, uint32_t sourceAddr, uint32_t* *destAddr, uint32_t length*)

Configures a transfer for the DMA.

Parameters

<i>chn</i>	Memory pointing to the DMA channel.
<i>type</i>	Transfer type.
<i>size</i>	Size to be transferred on each DMA write/read. Source/Dest share the same write/read size.
<i>sourceAddr</i>	Source address.
<i>destAddr</i>	Destination address.
<i>length</i>	Bytes to be transferred.

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.11 **dma_status_t DMA_DRV_ConfigChanLink (dma_channel_t * *chn*, dma_channel_link_config_t * *link_config*)**

Parameters

<i>chn</i>	Memory pointing to the DMA channel.
<i>link_config</i>	Configure of channel link in DMA.

Returns

If successful, returns the kStatus_DMA_Success. Otherwise, it returns an error.

8.4.5.12 **void DMA_DRV_IRQHandler (uint32_t *channel*)**

Parameters

<i>channel</i>	DMA channel number.
----------------	---------------------

8.4.6 Variable Documentation

8.4.6.1 **const uint32_t g_dmaRegBaseAddr[HW_DMA_INSTANCE_COUNT]**

8.4.6.2 **const uint32_t g_dmamuxRegBaseAddr[HW_DMAMUX_INSTANCE_COUNT]**

8.4.6.3 **const IRQn_Type g_dmalrqld[HW_DMA_INSTANCE_COUNT][FSL_FEATURE_DMA_DMAMUX_CHANNELS]**

8.5 DMA request

This section provides the DMA request resource.

8.5.1 DMA Initialization

To initialize the DMA module, call the `dma_init()` function. Configuration data structure does not need to be passed. This function enables the DMA module and clock automatically.

8.5.2 DMA Channel concept

DMA module consists of many channels. The DMA Peripheral driver is designed based on the channel concept. All tasks should start by requesting a DMA channel and end by freeing a DMA channel. By getting a channel allocated, the user can configure and run operations on the DMA module. If a channel is not allocated, a system error may occur.

8.5.3 DMA request concept

DMA request triggers a DMA transfer. The DMA request table is available in the device configuration chapters in a Reference Manual.

8.5.4 DMA Memory allocation

DMA peripheral driver does not allocate memory dynamically. The user must provide the allocated memory pointer for the driver and ensure that the memory is valid. Otherwise, a system error occurs. The user needs to provide the memory for the `[dma_channel_t]`. The driver must store the status data for every channel and the `dma_channel_t` is designed for this purpose.

8.5.5 DMA Call diagram

To use the DMA driver, follow these steps:

1. Initialize the DMA module: `dma_init()`.
2. Request a DMA channel: `dma_request_channel()`.
3. Configure the TCD: `dma_config_transfer()`.
4. Register callback function: `dma_register_callback()`.
5. Start the DMA channel: `dma_start_channel()`.
6. [OPTION] Stop the DMA channel: `dma_stop_channel()`.
7. Free the DMA channel: `dma_free_channel()`.

This example shows how to initialize and configure a memory-to-memory transfer:

```
uint32_t j, temp;
dma_channel_t chan_handler;
uint8_t *srcAddr, *destAddr;
fsl_rtos_status syncStatus;

srcAddr = malloc(kDmaTestBufferSize);
destAddr = malloc(kDmaTestBufferSize);
if (((uint32_t)srcAddr == 0x0U) & ((uint32_t)destAddr == 0x0U))
{
    printf("Fail to allocate memory for test! \r\n");
    goto error;
}

// Init the memory buffer. //
for (j = 0; j < kDmaTestBufferSize; j++)
{
    srcAddr[j] = j;
    destAddr[j] = 0;
}

temp = dma_request_channel(channel, kDmaRequestMux0AlwaysOn62, &chan_handler);
if (temp != channel)
{
    printf("Failed to request channel %d !\r\n", channel);
    goto error;
}

dma_config_transfer(&chan_handler, kDmaMemoryToMemory,
                    0x1U,
                    (uint32_t)srcAddr, (uint32_t)destAddr,
                    kDmaTestBufferSize);

dma_register_callback(&chan_handler, test_callback, &chan_handler);

dma_start_channel(&chan_handler);
//Wait until channel complete...
dma_stop_channel(&chan_handler);
dma_free_channel(&chan_handler);
```

DMA request

Chapter 9

Serial Peripheral Interface (DSPI)

9.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Serial Peripheral Interface (DSPI) block of Kinetis K- and V-series devices. For L-series Kinetis devices, refer to the SPI module driver.

Modules

- DSPI HAL driver
- DSPI Master Driver
- DSPI Shared IRQ Driver
- DSPI Slave Driver

9.2 DSPI HAL driver

9.2.1 Overview

This section describes the programming interface of the DSPI HAL driver.

Files

- file [fsl_dspi_hal.h](#)

Data Structures

- struct [dsPIC32Mx_DSPI_data_format_config_t](#)
DSPI data format settings configuration structure. [More...](#)
- struct [dsPIC32Mx_DSPI_slave_config_t](#)
DSPI hardware configuration settings for slave mode. [More...](#)
- struct [dsPIC32Mx_DSPI_baud_rate_divisors_t](#)
DSPI baud rate divisors settings configuration structure. [More...](#)
- struct [dsPIC32Mx_DSPI_command_config_t](#)
DSPI command and data configuration structure. [More...](#)

Enumerations

- enum [dsPIC32Mx_DSPI_status_t](#) { ,
 [kStatus_DSPI_SlaveTxUnderrun](#),
 [kStatus_DSPI_SlaveRxOverrun](#),
 [kStatus_DSPI_Timeout](#),
 [kStatus_DSPI_Busy](#),
 [kStatus_DSPI_NoTransferInProgress](#),
 [kStatus_DSPI_InvalidBitCount](#),
 [kStatus_DSPI_InvalidInstanceNumber](#),
 [kStatus_DSPI_OutOfRange](#),
 [kStatus_DSPI_InvalidParameter](#),
 [kStatus_DSPI_NonInit](#),
 [kStatus_DSPI_Initialized](#),
 [kStatus_DSPI_DMAChannelInvalid](#),
 [kStatus_DSPI_Error](#) }
Error codes for the DSPI driver.
- enum [dsPIC32Mx_DSPI_master_slave_mode_t](#) {
 [kDspiMaster](#) = 1,
 [kDspiSlave](#) = 0 }
DSPI master or slave configuration.
- enum [dsPIC32Mx_DSPI_clock_polarity_t](#) {
 [kDspiClockPolarity_ActiveHigh](#) = 0,
 [kDspiClockPolarity_ActiveLow](#) = 1 }

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

 `kDspiClockPhase_FirstEdge` = 0,

 `kDspiClockPhase_SecondEdge` = 1 }

 DSPI clock phase configuration for a given CTAR.
- enum `dspi_shift_direction_t` {

 `kDspiMsbFirst` = 0,

 `kDspiLsbFirst` = 1 }

 DSPI data shifter direction options for a given CTAR.
- enum `dspi_ctar_selection_t` {

 `kDspiCtar0` = 0,

 `kDspiCtar1` = 1 }

 DSPI Clock and Transfer Attributes Register (CTAR) selection.
- enum `dspi_pcs_polarity_config_t` {

 `kDspiPcs_ActiveHigh` = 0,

 `kDspiPcs_ActiveLow` = 1 }

 DSPI Peripheral Chip Select (PCS) Polarity configuration.
- enum `dspi_which_pcs_config_t` {

 `kDspiPcs0` = 1 << 0,

 `kDspiPcs1` = 1 << 1,

 `kDspiPcs2` = 1 << 2,

 `kDspiPcs3` = 1 << 3,

 `kDspiPcs4` = 1 << 4,

 `kDspiPcs5` = 1 << 5 }

 DSPI Peripheral Chip Select (PCS) configuration (which PCS to configure)
- enum `dspi_master_sample_point_t` {

 `kDspiSckToSin_0Clock` = 0,

 `kDspiSckToSin_1Clock` = 1,

 `kDspiSckToSin_2Clock` = 2 }

 DSPI Sample Point: Controls when the DSPI master samples SIN in Modified Transfer Format.
- enum `dspi_fifo_t` {

 `kDspiTxFifo` = 0,

 `kDspiRxFifo` = 1 }

 DSPI FIFO selects.
- enum `dspi_dma_or_int_mode_t` {

 `kDspiGenerateIntReq` = 0,

 `kDspiGenerateDmaReq` = 1 }

 DSPI Tx FIFO Fill and Rx FIFO Drain DMA or Interrupt configuration.
- enum `dspi_status_and_interrupt_request_t` {

 `kDspiTxComplete` = BP_SPI_RSER_TCF_RE,

 `kDspiTxAndRxStatus` = BP_SPI_SR_TXRXS,

 `kDspiEndOfQueue` = BP_SPI_RSER_EOQF_RE,

 `kDspiTxFifoUnderflow` = BP_SPI_RSER_TFUF_RE,

 `kDspiTxFifoFillRequest` = BP_SPI_RSER_TFFF_RE,

 `kDspiRxFifoOverflow` = BP_SPI_RSER_RFOF_RE,

 `kDspiRxFifoDrainRequest` = BP_SPI_RSER_RFDF_RE }
- DSPI status flags and interrupt request enable.

DSPI HAL driver

- enum `dspi_fifo_counter_pointer_t` {
 `kDspiRxFifoPointer` = BP_SPI_SR_POPNXTPTR,
 `kDspiRxFifoCounter` = BP_SPI_SR_RXCTR,
 `kDspiTxFifoPointer` = BP_SPI_SR_TXNXTPTR,
 `kDspiTxFifoCounter` = BP_SPI_SR_TXCTR }
 DSPI FIFO counter or pointer defines based on bit positions.
- enum `dspi_delay_type_t` {
 `kDspiPcsToSck` = 1,
 `kDspiLastSckToPcs` = 2,
 `kDspiAfterTransfer` = 3 }
 DSPI delay type selection.

Configuration

- void `DSPI_HAL_Init` (uint32_t baseAddr)
 Restores the DSPI to reset the configuration.
- static void `DSPI_HAL_Enable` (uint32_t baseAddr)
 Enables the DSPI peripheral and sets the MCR MDIS to 0.
- static void `DSPI_HAL_Disable` (uint32_t baseAddr)
 Disables the DSPI peripheral, sets MCR MDIS to 1.
- uint32_t `DSPI_HAL_SetBaudRate` (uint32_t baseAddr, `dspi_ctar_selection_t` whichCtar, uint32_t bitsPerSec, uint32_t sourceClockInHz)
 Sets the DSPI baud rate in bits per second.
- void `DSPI_HAL_SetBaudDivisors` (uint32_t baseAddr, `dspi_ctar_selection_t` whichCtar, const `dspi_baud_rate_divisors_t` *divisors)
 Configures the baud rate divisors manually.
- static void `DSPI_HAL_SetMasterSlaveMode` (uint32_t baseAddr, `dspi_master_slave_mode_t` mode)
 Configures the DSPI for master or slave.
- static bool `DSPI_HAL_IsMaster` (uint32_t baseAddr)
 Returns whether the DSPI module is in master mode.
- static void `DSPI_HAL_SetContinuousSckCmd` (uint32_t baseAddr, bool enable)
 Configures the DSPI for the continuous SCK operation.
- static void `DSPI_HAL_SetModifiedTimingFormatCmd` (uint32_t baseAddr, bool enable)
 Configures the DSPI to enable modified timing format.
- static void `DSPI_HAL_SetRxFifoOverwriteCmd` (uint32_t baseAddr, bool enable)
 Configures the DSPI received FIFO overflow overwrite enable.
- void `DSPI_HAL_SetPcsPolarityMode` (uint32_t baseAddr, `dspi_which_pcs_config_t` pcs, `dspi_pcs_polarity_config_t` activeLowOrHigh)
 Configures the DSPI peripheral chip select polarity.
- void `DSPI_HAL_SetFifoCmd` (uint32_t baseAddr, bool enableTxFifo, bool enableRxFifo)
 Enables (or disables) the DSPI FIFOs.
- void `DSPI_HAL_SetFlushFifoCmd` (uint32_t baseAddr, bool enableFlushTxFifo, bool enableFlushRxFifo)
 Flushes the DSPI FIFOs.
- static void `DSPI_HAL_SetDatainSamplepointMode` (uint32_t baseAddr, `dspi_master_sample_point_t` samplePnt)
 Configures the time when the DSPI master samples SIN in the Modified Transfer Format.

- static void **DSPI_HAL_StartTransfer** (uint32_t baseAddr)
Starts the DSPI transfers, clears HALT bit in MCR.
- static void **DSPI_HAL_StopTransfer** (uint32_t baseAddr)
Stops (halts) DSPI transfers, sets HALT bit in MCR.
- **dspi_status_t DSPI_HAL_SetDataFormat** (uint32_t baseAddr, **dspi_ctar_selection_t** whichCtar, const **dspi_data_format_config_t** *config)
Configures the data format for a particular CTAR.
- void **DSPI_HAL_SetDelay** (uint32_t baseAddr, **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_HAL_CalculateDelay** (uint32_t baseAddr, **dspi_ctar_selection_t** whichCtar, **dspi_delay_type_t** whichDelay, uint32_t sourceClockInHz, uint32_t delayInNanoSec)
Calculates the delay prescaler and scaler based on the desired delay input in nanoseconds.
- static uint32_t **DSPI_HAL_GetMasterPushrRegAddr** (uint32_t baseAddr)
Gets the DSPI master PUSHR data register address for DMA operation.
- static uint32_t **DSPI_HAL_GetSlavePushrRegAddr** (uint32_t baseAddr)
Gets the DSPI slave PUSHR data register address for DMA operation.
- static uint32_t **DSPI_HAL_GetPoprRegAddr** (uint32_t baseAddr)
Gets the DSPI POPR data register address for DMA operation.

Low power

- static void **DSPI_HAL_SetDozemodeCmd** (uint32_t baseAddr, bool enable)
Configures the DSPI operation during doze mode.

Interrupts

- void **DSPI_HAL_SetTxFifoFillDmaIntMode** (uint32_t baseAddr, **dspi_dma_or_int_mode_t** mode, bool enable)
Configures the DSPI Tx FIFO fill request to generate DMA or interrupt requests.
- void **DSPI_HAL_SetRxFifoDrainDmaIntMode** (uint32_t baseAddr, **dspi_dma_or_int_mode_t** mode, bool enable)
Configures the DSPI Rx FIFO Drain request to generate DMA or interrupt requests.
- void **DSPI_HAL_SetIntMode** (uint32_t baseAddr, **dspi_status_and_interrupt_request_t** interruptSrc, bool enable)
Configures the DSPI interrupts.
- static bool **DSPI_HAL_GetIntMode** (uint32_t baseAddr, **dspi_status_and_interrupt_request_t** interruptSrc)
Gets DSPI interrupt configuration, returns if interrupt request is enabled or disabled.

Status

- static bool **DSPI_HAL_GetStatusFlag** (uint32_t baseAddr, **dspi_status_and_interrupt_request_t** statusFlag)
Gets the DSPI status flag state.

DSPI HAL driver

- static void **DSPI_HAL_ClearStatusFlag** (uint32_t baseAddr, **dspi_status_and_interrupt_request_t** statusFlag)
Clears the DSPI status flag.
- static uint32_t **DSPI_HAL_GetFifoCountOrPtr** (uint32_t baseAddr, **dspi_fifo_counter_pointer_t** desiredParameter)
Gets the DSPI FIFO counter or pointer.

Data transfer

- static uint32_t **DSPI_HAL_ReadData** (uint32_t baseAddr)
Reads data from the data buffer.
- static void **DSPI_HAL_WriteDataSlavemode** (uint32_t baseAddr, uint32_t data)
Writes data into the data buffer, slave mode.
- void **DSPI_HAL_WriteDataSlavemodeBlocking** (uint32_t baseAddr, uint32_t data)
Writes data into the data buffer, slave mode and waits till data was transmitted and return .
- void **DSPI_HAL_WriteDataMastermode** (uint32_t baseAddr, **dspi_command_config_t** *command, uint16_t data)
Writes data into the data buffer, master mode.
- void **DSPI_HAL_WriteDataMastermodeBlocking** (uint32_t baseAddr, **dspi_command_config_t** *command, uint16_t data)
Writes data into the data buffer, master mode and waits till complete to return.
- static void **DSPI_HAL_WriteCmdDataMastermode** (uint32_t baseAddr, uint32_t data)
Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer, master mode.
- void **DSPI_HAL_WriteCmdDataMastermodeBlocking** (uint32_t baseAddr, 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 uint32_t **DSPI_HAL_GetTransferCount** (uint32_t baseAddr)
Gets the transfer count.
- static void **DSPI_HAL_PresetTransferCount** (uint32_t baseAddr, uint16_t presetValue)
Pre-sets the transfer count.
- uint32_t **DSPI_HAL_GetFormattedCommand** (uint32_t baseAddr, **dspi_command_config_t** *command)
Returns the DSPI command word formatted to the PUSHR data register bit field.

Debug

- uint32_t **DSPI_HAL_GetFifoData** (uint32_t baseAddr, **dspi_fifo_t** whichFifo, uint32_t whichFifoEntry)
Reads FIFO registers for debug purposes.
- static void **DSPI_HAL_SetHaltInDebugmodeCmd** (uint32_t baseAddr, bool enable)
Configures the DSPI to halt during debug mode.

9.2.2 Data Structure Documentation

9.2.2.1 struct dsPIC33FJ256GP202/dspi_data_format_config_t

This structure contains the data format settings. These settings apply to a specific CTARn register, which the user must provide in this structure.

Data Fields

- `uint32_t bitsPerFrame`
Bits per frame, minimum 4, maximum 16 (master), 32 (slave)
- `dsPIC33FJ256GP202/dspi_clock_polarity_t clkPolarity`
Active high or low clock polarity.
- `dsPIC33FJ256GP202/dspi_clock_phase_t clkPhase`
Clock phase setting to change and capture data.
- `dsPIC33FJ256GP202/dspi_shift_direction_t direction`
MSB or LSB data shift direction This setting relevant only in master mode and can be ignored in slave mode.

9.2.2.2 struct dsPIC33FJ256GP202/dspi_slave_config_t

Use an instance of this structure with the DSPI_HAL_SlaveInit() to configure the most common settings of the DSPI peripheral in slave mode with a single function call.

Data Fields

- `bool isEnabled`
Set to true to enable the DSPI peripheral.
- `dsPIC33FJ256GP202/dspi_data_format_config_t dataConfig`
Data format configuration structure.
- `bool isTxFifoDisabled`
Disable(1) or Enable(0) Tx FIFO.
- `bool isRxFifoDisabled`
Disable(1) or Enable(0) Rx FIFO.

9.2.2.2.0.14 Field Documentation

9.2.2.2.0.14.1 bool dsPIC33FJ256GP202/dspi_slave_config_t::isEnabled

9.2.2.3 struct dsPIC33FJ256GP202/dspi_baud_rate_divisors_t

Note: These settings are relevant only in master mode. This structure contains the baud rate divisor settings, which provides the user with the option to explicitly set these baud rate divisors. In addition, the user must also set the CTARn register with the divisor settings.

Data Fields

- bool `doubleBaudRate`
Double Baud rate parameter setting.
- `uint32_t prescaleDivisor`
Baud Rate Pre-scalar parameter setting.
- `uint32_t baudRateDivisor`
Baud Rate scaler parameter setting.

9.2.2.4 `struct dspi_command_config_t`

Note: This structure is used with the PUSHR register, which provides the means to write to the Tx FIFO. Data written to this register is transferred to the Tx FIFO. Eight or sixteen-bit write accesses to the PUSHR transfer all 32 register bits to the Tx FIFO. The register structure is different in master and slave modes. In master mode, the register provides 16-bit command and 16-bit data to the Tx FIFO. In slave mode all 32 register bits can be used as data, supporting up to 32-bit SPI frame operation.

Data Fields

- bool `isChipSelectContinuous`
Option to enable the continuous assertion of chip select between transfers.
- `dspi_ctar_selection_t whichCtar`
The desired Clock and Transfer Attributes Register (CTAR) to use for CTAS.
- `dspi_which_pcs_config_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 SPI_TCNT field; cleared before transmission starts.

9.2.3 Enumeration Type Documentation

9.2.3.1 `enum dspi_status_t`

Enumerator

`kStatus_DSPI_SlaveTxUnderrun` DSPI Slave Tx Under run error.

`kStatus_DSPI_SlaveRxOverrun` DSPI Slave Rx Overrun error.

`kStatus_DSPI_Timeout` DSPI transfer timed out.

`kStatus_DSPI_Busy` DSPI instance is already busy performing a transfer.

`kStatus_DSPI_NoTransferInProgress` Attempt to abort a transfer when no transfer was in progress.

`kStatus_DSPI_InvalidBitCount` bits-per-frame value not valid

`kStatus_DSPI_InvalidInstanceNumber` DSPI instance number does not match current count.

`kStatus_DSPI_OutOfRange` DSPI out-of-range error used in slave callback.

`kStatus_DSPI_InvalidParameter` DSPI invalid parameter error.

kStatus_DSPI_NonInit DSPI driver does not initialize, not ready.

kStatus_DSPI_Initialized DSPI driver has initialized, could not initialize again.

kStatus_DSPI_DMACHannelInvalid DSPI driver could not request DMA channel(s)

kStatus_DSPI_Error DSPI driver error.

9.2.3.2 enum dspi_master_slave_mode_t

Enumerator

kDspiMaster DSPI peripheral operates in master mode.

kDspiSlave DSPI peripheral operates in slave mode.

9.2.3.3 enum dspi_clock_polarity_t

Enumerator

kDspiClockPolarity_ActiveHigh Active-high DSPI clock (idles low)

kDspiClockPolarity_ActiveLow Active-low DSPI clock (idles high)

9.2.3.4 enum dspi_clock_phase_t

Enumerator

kDspiClockPhase_FirstEdge Data is captured on the leading edge of the SCK and changed on the following edge.

kDspiClockPhase_SecondEdge Data is changed on the leading edge of the SCK and captured on the following edge.

9.2.3.5 enum dspi_shift_direction_t

Enumerator

kDspiMsbFirst Data transfers start with most significant bit.

kDspiLsbFirst Data transfers start with least significant bit.

9.2.3.6 enum dspi_ctar_selection_t

Enumerator

kDspiCtar0 CTAR0 selection option for master or slave mode.

kDspiCtar1 CTAR1 selection option for master mode only.

DSPI HAL driver

9.2.3.7 enum dspi_pcs_polarity_config_t

Enumerator

kDspiPcs_ActiveHigh PCS Active High (idles low)

kDspiPcs_ActiveLow PCS Active Low (idles high)

9.2.3.8 enum dspi_which_pcs_config_t

Enumerator

kDspiPcs0 PCS[0].

kDspiPcs1 PCS[1].

kDspiPcs2 PCS[2].

kDspiPcs3 PCS[3].

kDspiPcs4 PCS[4].

kDspiPcs5 PCS[5].

9.2.3.9 enum dspi_master_sample_point_t

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

Enumerator

kDspiSckToSin_0Clock 0 system clocks between SCK edge and SIN sample

kDspiSckToSin_1Clock 1 system clock between SCK edge and SIN sample

kDspiSckToSin_2Clock 2 system clocks between SCK edge and SIN sample

9.2.3.10 enum dspi_fifo_t

Enumerator

kDspiTxFifo DSPI Tx FIFO.

kDspiRxFifo DSPI Rx FIFO.

9.2.3.11 enum dspi_dma_or_int_mode_t

Enumerator

kDspiGenerateIntReq Desired flag generates an Interrupt request.

kDspiGenerateDmaReq Desired flag generates a DMA request.

9.2.3.12 enum dspi_status_and_interrupt_request_t

Enumerator

- kDspiTxComplete* TCF status/interrupt enable.
- kDspiTxAndRxStatus* TXRXS status only, no interrupt.
- kDspiEndOfQueue* EOQF status/interrupt enable.
- kDspiTxFifoUnderflow* TFUF status/interrupt enable.
- kDspiTxFifoFillRequest* TFFF status/interrupt enable.
- kDspiRxFifoOverflow* RFOF status/interrupt enable.
- kDspiRxFifoDrainRequest* RFDF status/interrupt enable.

9.2.3.13 enum dspi_fifo_counter_pointer_t

Enumerator

- kDspiRxFifoPointer* Rx FIFO pointer.
- kDspiRxFifoCounter* Rx FIFO counter.
- kDspiTxFifoPointer* Tx FIFO pointer.
- kDspiTxFifoCounter* Tx FIFO counter.

9.2.3.14 enum dspi_delay_type_t

Enumerator

- kDspiPcsToSck* PCS-to-SCK delay.
- kDspiLastSckToPcs* Last SCK edge to PCS delay.
- kDspiAfterTransfer* Delay between transfers.

9.2.4 Function Documentation

9.2.4.1 void DSPI_HAL_Init(uint32_t baseAddr)

This function basically resets all of the DSPI registers to their default setting including disabling the module.

Parameters

DSPI HAL driver

<i>baseAddr</i>	Module base address
-----------------	---------------------

9.2.4.2 static void DSPI_HAL_Enable(uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

9.2.4.3 static void DSPI_HAL_Disable(uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

9.2.4.4 uint32_t DSPI_HAL_SetBaudRate(uint32_t *baseAddr*, dspi_ctar_selection_t *whichCtar*, uint32_t *bitsPerSec*, uint32_t *sourceClockInHz*)

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate, and returns the calculated baud rate in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

<i>baseAddr</i>	Module base address
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of the type dspi_ctar_selection_t
<i>bitsPerSec</i>	The desired baud rate in bits per second
<i>sourceClockInHz</i>	Module source input clock in Hertz

Returns

The actual calculated baud rate

9.2.4.5 void DSPI_HAL_SetBaudDivisors(uint32_t *baseAddr*, dspi_ctar_selection_t *whichCtar*, const dspi_baud_rate_divisors_t * *divisors*)

This function allows the caller to manually set the baud rate divisors in the event that these dividers are known and the caller does not wish to call the DSPI_HAL_SetBaudRate function.

Parameters

<i>baseAddr</i>	Module base address
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type <code>dspi_ctar_selection_t</code>
<i>divisors</i>	Pointer to a structure containing the user defined baud rate divisor settings

9.2.4.6 static void DSPI_HAL_SetMasterSlaveMode (`uint32_t baseAddr,` `dspi_master_slave_mode_t mode`) [inline], [static]

Parameters

<i>baseAddr</i>	Module base address
<i>mode</i>	Mode setting (master or slave) of type <code>dspi_master_slave_mode_t</code>

9.2.4.7 static bool DSPI_HAL_IsMaster (`uint32_t baseAddr`) [inline], [static]

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

Returns

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

9.2.4.8 static void DSPI_HAL_SetContinuousSckCmd (`uint32_t baseAddr, bool enable`) [inline], [static]

Parameters

<i>baseAddr</i>	Module base address
<i>enable</i>	Enables (true) or disables(false) continuous SCK operation.

9.2.4.9 static void DSPI_HAL_SetModifiedTimingFormatCmd (`uint32_t baseAddr, bool enable`) [inline], [static]

DSPI HAL driver

Parameters

<i>baseAddr</i>	Module base address
<i>enable</i>	Enables (true) or disables(false) modified timing format.

9.2.4.10 static void DSPI_HAL_SetRx_fifoOverwriteCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

When enabled, this function allows incoming receive data to overwrite the existing data in the receive shift register when the Rx FIFO is full. Otherwise when disabled, the incoming data is ignored when the RX FIFO is full.

Parameters

<i>baseAddr</i>	Module base address.
<i>enable</i>	If enabled (true), allows incoming data to overwrite Rx FIFO contents when full, else incoming data is ignored.

9.2.4.11 void DSPI_HAL_SetPcsPolarityMode (uint32_t *baseAddr*, dspi_which_pcs_config_t *pcs*, dspi_pcs_polarity_config_t *activeLowOrHigh*)

This function takes in the desired peripheral chip select (PCS) and it's corresponding desired polarity and configures the PCS signal to operate with the desired characteristic.

Parameters

<i>baseAddr</i>	Module base address
<i>pcs</i>	The particular peripheral chip select (parameter value is of type dspi_which_pcs_config_t) for which we wish to apply the active high or active low characteristic.
<i>activeLowOrHigh</i>	The setting for either "active high, inactive low (0)" or "active low, inactive high(1)" of type dspi_pcs_polarity_config_t.

9.2.4.12 void DSPI_HAL_SetFifoCmd (uint32_t *baseAddr*, bool *enableTxFifo*, bool *enableRxFifo*)

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

Parameters

<i>baseAddr</i>	Module instance number
<i>enableTxFifo</i>	Disables (false) the TX FIFO, else enables (true) the TX FIFO
<i>enableRxFifo</i>	Disables (false) the RX FIFO, else enables (true) the RX FIFO

9.2.4.13 void DSPI_HAL_SetFlushFifoCmd (*uint32_t baseAddr*, *bool enableFlushTxFifo*, *bool enableFlushRxFifo*)

Parameters

<i>baseAddr</i>	Module base address
<i>enableFlushTx-Fifo</i>	Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO
<i>enableFlush-RxFifo</i>	Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO

9.2.4.14 static void DSPI_HAL_SetDatainSamplepointMode (*uint32_t baseAddr*, *dspi_master_sample_point_t samplePnt*) [inline], [static]

This function controls when the DSPI master samples SIN (data in) in the Modified Transfer Format. Note that this is valid only when the CPHA bit in the CTAR register is 0.

Parameters

<i>baseAddr</i>	Module base address
<i>samplePnt</i>	selects when the data in (SIN) is sampled, of type <i>dspi_master_sample_point_t</i> . This value selects either 0, 1, or 2 system clocks between the SCK edge and the SIN (data in) sample.

9.2.4.15 static void DSPI_HAL_StartTransfer (*uint32_t baseAddr*) [inline], [static]

This function call called whenever the module is ready to begin data transfers in either master or slave mode.

DSPI HAL driver

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

9.2.4.16 static void DSPI_HAL_StopTransfer (uint32_t *baseAddr*) [inline], [static]

This function call stops data transfers in either master or slave mode.

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

9.2.4.17 dspi_status_t DSPI_HAL_SetDataFormat (uint32_t *baseAddr*, dspi_ctar_selection_t *whichCtar*, const dspi_data_format_config_t * *config*)

This function configures the bits-per-frame, polarity, phase, and shift direction for a particular CTAR. An example use case is as follows:

```
dspi_data_format_config_t dataFormat;
dataFormat.bitsPerFrame = 16;
dataFormat.clkPolarity = kDspiClockPolarity_ActiveLow;
dataFormat.clkPhase = kDspiClockPhase_FirstEdge;
dataFormat.direction = kDspiMsbFirst;
DSPI_HAL_SetDataFormat(instance, kDspiCtar0, &dataFormat);
```

Parameters

<i>baseAddr</i>	Module base address
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_selection_t.
<i>config</i>	Pointer to structure containing user defined data format configuration settings.

Returns

An error code or kStatus_DSPI_Success

9.2.4.18 void DSPI_HAL_SetDelay (uint32_t *baseAddr*, 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 type `dspi_delay_type_t`.

The user passes which delay they want to configure along with the prescaler and scaler value. This allows the user to directly set the prescaler/scaler values if they have pre-calculated them or if they simply wish to manually increment either value.

Parameters

<i>baseAddr</i>	Module base address
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type <code>dspi_ctar_selection_t</code> .
<i>prescaler</i>	The prescaler delay value (can be an integer 0, 1, 2, or 3).
<i>scaler</i>	The scaler delay value (can be any integer between 0 to 15).
<i>whichDelay</i>	The desired delay to configure, must be of type <code>dspi_delay_type_t</code>

9.2.4.19 `uint32_t DSPI_HAL_CalculateDelay (uint32_t baseAddr, dspi_ctar_selection_t whichCtar, dspi_delay_type_t whichDelay, uint32_t sourceClockInHz, uint32_t delayInNanoSec)`

This function calculates the values for: 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 type `dspi_delay_type_t`.

The user passes which delay they want to configure along with the desired delay value in nano-seconds. The function calculates the values needed for the prescaler and scaler and returning the actual 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 will be returned. It is to the higher level peripheral driver to alert the user of an out of range delay input.

Parameters

<i>baseAddr</i>	Module base address
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type <code>dspi_ctar_selection_t</code> .
<i>whichDelay</i>	The desired delay to configure, must be of type <code>dspi_delay_type_t</code>

DSPI HAL driver

<i>sourceClockIn-Hz</i>	Module source input clock in Hertz
<i>delayInNano-Sec</i>	The desired delay value in nano-seconds.

Returns

The actual calculated delay value.

9.2.4.20 static uint32_t DSPI_HAL_GetMasterPushrRegAddr (*uint32_t baseAddr*) [**inline**], [**static**]

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

Parameters

<i>baseAddr</i>	Module base address.
-----------------	----------------------

Returns

The DSPI master PUSHR data register address.

9.2.4.21 static uint32_t DSPI_HAL_GetSlavePushrRegAddr (*uint32_t baseAddr*) [**inline**], [**static**]

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

Parameters

<i>baseAddr</i>	Module base address.
-----------------	----------------------

Returns

The DSPI slave PUSHR data register address.

9.2.4.22 static uint32_t DSPI_HAL_GetPoprRegAddr (*uint32_t baseAddr*) [**inline**], [**static**]

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

Parameters

<i>baseAddr</i>	Module base address.
-----------------	----------------------

Returns

The DSPI POPR data register address.

9.2.4.23 static void DSPI_HAL_SetDozemodeCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function provides support for an externally controlled doze mode, power-saving, mechanism. When disabled, the doze mode has no effect on the DSPI, and when enabled, the Doze mode disables the DSPI.

Parameters

<i>baseAddr</i>	Module base address
<i>enable</i>	If disabled (false), the doze mode has no effect on the DSPI, if enabled (true), the doze mode disables the DSPI.

9.2.4.24 void DSPI_HAL_SetTxFifoFillDmaIntMode (uint32_t *baseAddr*, dspi_dma_or_int_mode_t *mode*, bool *enable*)

This function configures the DSPI Tx FIFO Fill flag to generate either an interrupt or DMA request. The user passes in which request they'd like to generate of type `dspi_dma_or_int_mode_t` and whether or not they wish to enable this request. Note, when disabling the request, the request type is don't care.

```
DSPI_HAL_SetTxFifoFillDmaIntMode(baseAddr,
    kDspiGenerateDmaReq, true); <- to enable DMA
DSPI_HAL_SetTxFifoFillDmaIntMode(baseAddr,
    kDspiGenerateIntReq, true); <- to enable Interrupt
DSPI_HAL_SetTxFifoFillDmaIntMode(baseAddr,
    kDspiGenerateIntReq, false); <- to disable
```

Parameters

<i>baseAddr</i>	Module base address
<i>mode</i>	Configures the DSPI Tx FIFO Fill to generate an interrupt or DMA request

DSPI HAL driver

<i>enable</i>	Enable (true) or disable (false) the DSPI Tx FIFO Fill flag to generate requests
---------------	--

9.2.4.25 void DSPI_HAL_SetRx_fifoDrainDmaIntMode (uint32_t *baseAddr*, dspi_dma_or_int_mode_t *mode*, bool *enable*)

This function configures the DSPI Rx FIFO Drain flag to generate either an interrupt or a DMA request. The user passes in which request they'd like to generate of type *dspi_dma_or_int_mode_t* and whether or not they wish to enable this request. Note, when disabling the request, the request type is don't care.

```
DSPI_HAL_SetRx_fifoDrainDmaIntMode(baseAddr,  
    kDspiGenerateDmaReq, true); <- to enable DMA  
DSPI_HAL_SetRx_fifoDrainDmaIntMode(baseAddr,  
    kDspiGenerateIntReq, true); <- to enable Interrupt  
DSPI_HAL_SetRx_fifoDrainDmaIntMode(baseAddr,  
    kDspiGenerateIntReq, false); <- to disable
```

Parameters

<i>baseAddr</i>	Module base address
<i>mode</i>	Configures the Rx FIFO Drain to generate an interrupt or DMA request
<i>enable</i>	Enable (true) or disable (false) the Rx FIFO Drain flag to generate requests

9.2.4.26 void DSPI_HAL_SetIntMode (uint32_t *baseAddr*, dspi_status_and_interrupt_request_t *interruptSrc*, bool *enable*)

This function configures the various interrupt sources of the DSPI. The parameters are *baseAddr*, *interrupt source*, and *enable/disable* setting. The *interrupt source* is a typedef enumeration whose value is the bit position of the interrupt source setting within the RSER register. In the DSPI, all interrupt configuration settings are in one register. The typedef enum equates each interrupt source to the bit position defined in the device header file. The function uses these bit positions in its algorithm to enable/disable the interrupt source, where *interrupt source* is the *dspi_status_and_interrupt_request_t* type. Note, for Tx FIFO Fill and Rx FIFO Drain requests, use the functions: DSPI_HAL_SetTxFifoFillDmaIntMode and DSPI_HAL_SetRx_fifoDrainDmaIntMode respectively as these requests can generate either an interrupt or DMA request.

```
DSPI_HAL_SetIntMode(baseAddr, kDspiTxComplete, true); <- example use-case
```

Parameters

<i>baseAddr</i>	Module base address
<i>interruptSrc</i>	The interrupt source, of type dspi_status_and_interrupt_request_t
<i>enable</i>	Enable (true) or disable (false) the interrupt source to generate requests

9.2.4.27 static bool DSPI_HAL_GetIntMode (uint32_t *baseAddr*, dspi_status_and_interrupt_request_t *interruptSrc*) [inline], [static]

This function returns the requested interrupt source setting (enabled or disabled, of type bool). The parameters to pass in are baseAddr and interrupt source. It utilizes the same enumeration definitions for the interrupt sources as described in the "interrupt configuration" function. The function uses these bit positions in its algorithm to obtain the desired interrupt source setting. Note, for Tx FIFO Fill and Rx FIFO Drain requests, this returns whether or not their requests are enabled.

```
getInterruptSetting = DSPI_HAL_GetIntMode(baseAddr,
                                         kDspiTxComplete);
```

Parameters

<i>baseAddr</i>	Module base address
<i>interruptSrc</i>	The interrupt source, of type dspi_status_and_interrupt_request_t

Returns

Configuration of interrupt request: enable (true) or disable (false).

9.2.4.28 static bool DSPI_HAL_GetStatusFlag (uint32_t *baseAddr*, dspi_status_and_interrupt_request_t *statusFlag*) [inline], [static]

The status flag is defined in the same enumeration as the interrupt source enable because the bit position of the interrupt source and corresponding status flag are the same in the RSER and SR registers. The function uses these bit positions in its algorithm to obtain the desired flag state, similar to the dspi_get_interrupt_config function.

```
getStatus = DSPI_HAL_GetStatusFlag(baseAddr,
                                    kDspiTxComplete);
```

DSPI HAL driver

Parameters

<i>baseAddr</i>	Module base address
<i>statusFlag</i>	The status flag, of type <code>dspi_status_and_interrupt_request_t</code>

Returns

State of the status flag: asserted (true) or not-asserted (false)

9.2.4.29 static void DSPI_HAL_ClearStatusFlag (`uint32_t baseAddr,` `dspi_status_and_interrupt_request_t statusFlag`) [inline], [static]

This function clears the desired status bit by using a write-1-to-clear. The user passes in the baseAddr 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. Example usage:

```
DSPI_HAL_ClearStatusFlag(baseAddr, kDspiTxComplete);
```

Parameters

<i>baseAddr</i>	Module base address
<i>statusFlag</i>	The status flag, of type <code>dspi_status_and_interrupt_request_t</code>

9.2.4.30 static uint32_t DSPI_HAL_GetFifoCountOrPtr (`uint32_t baseAddr,` `dspi_fifo_counter_pointer_t desiredParameter`) [inline], [static]

This function returns the number of entries or the next pointer in the Tx or Rx FIFO. The parameters to pass in are the baseAddr and either the Tx or Rx FIFO counter or a pointer. The latter is an enumeration type defined as the bitmask of those particular bit fields found in the device header file. Example usage:

```
DSPI_HAL_GetFifoCountOrPtr(baseAddr,  
    kDspiRxFifoCounter);
```

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

<i>desired-Parameter</i>	Desired parameter to obtain, of type dspi_fifo_counter_pointer_t
--------------------------	--

Returns

The number of entries or the next pointer in the Tx or Rx FIFO

9.2.4.31 static uint32_t DSPI_HAL_ReadData (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

Returns

The data from the read data buffer

9.2.4.32 static void DSPI_HAL_WriteDataSlavemode (uint32_t *baseAddr*, uint32_t *data*) [inline], [static]

In slave mode, up to 32-bit words may be written.

Parameters

<i>baseAddr</i>	Module base address
<i>data</i>	The data to send

9.2.4.33 void DSPI_HAL_WriteDataSlavemodeBlocking (uint32_t *baseAddr*, uint32_t *data*)

In slave mode, up to 32-bit words may be written. Function firstly clear transmit complete flag then write data into data register, finally wait tills the data transmitted.

Parameters

DSPI HAL driver

<i>baseAddr</i>	Module base address
<i>data</i>	The data to send

9.2.4.34 void DSPI_HAL_WriteDataMastermode (uint32_t *baseAddr*, dspi_command_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: 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.isChipSelectContinuous = true;
commandConfig.whichCtar = kDspiCtar0;
commandConfig.whichPcs = kDspiPcs1;
commandConfig.clearTransferCount = false;
commandConfig.isEndOfQueue = false;
DSPI_HAL_WriteDataMastermode(baseAddr, &commandConfig, dataWord);
```

Parameters

<i>baseAddr</i>	Module base address
<i>command</i>	Pointer to command structure
<i>data</i>	The data word to be sent

9.2.4.35 void DSPI_HAL_WriteDataMastermodeBlocking (uint32_t *baseAddr*, dspi_command_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: 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.isChipSelectContinuous = true;
commandConfig.whichCtar = kDspiCtar0;
commandConfig.whichPcs = kDspiPcs1;
commandConfig.clearTransferCount = false;
commandConfig.isEndOfQueue = false;
DSPI_HAL_WriteDataMastermodeBlocking(baseAddr, &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 in order to transmit data (MCR[MDIS] & [HALT] = 0). Since the SPI is a synchronous protocol, receive data is available when transmit completes.

Parameters

<i>baseAddr</i>	Module base address
<i>command</i>	Pointer to command structure
<i>data</i>	The data word to be sent

9.2.4.36 static void DSPI_HAL_WriteCmdDataMastermode (uint32_t *baseAddr*, uint32_t *data*) [inline], [static]

In this function, the user must append the 16-bit data to the 16-bit command info then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data such as: 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_HAL_WriteCmdDataMastermode(baseAddr, dataWord);
```

Parameters

<i>baseAddr</i>	Module base address
<i>data</i>	The data word (command and data combined) to be sent

9.2.4.37 void DSPI_HAL_WriteCmdDataMastermodeBlocking (uint32_t *baseAddr*, uint32_t *data*)

In this function, the user must append the 16-bit data to the 16-bit command info then provide the total 32-bit word as the data to send. The command portion provides characteristics of the data such as: 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_HAL_WriteCmdDataMastermodeBlocking(baseAddr, dataWord);
```

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

DSPI HAL driver

Parameters

<i>baseAddr</i>	Module base address
<i>data</i>	The data word (command and data combined) to be sent

9.2.4.38 static uint32_t DSPI_HAL_GetTransferCount (uint32_t *baseAddr*) [inline], [static]

This function returns the current value of the DSPI Transfer Count Register.

Parameters

<i>baseAddr</i>	Module base address
-----------------	---------------------

Returns

The current transfer count

9.2.4.39 static void DSPI_HAL_PresetTransferCount (uint32_t *baseAddr*, uint16_t *presetValue*) [inline], [static]

This function allows the caller to pre-set the DSPI Transfer Count Register to a desired value up to 65535; Incrementing past this resets the counter back to 0.

Parameters

<i>baseAddr</i>	Module base address
<i>presetValue</i>	The desired pre-set value for the transfer counter

9.2.4.40 uint32_t DSPI_HAL_GetFormattedCommand (uint32_t *baseAddr*, dspi_command_config_t * *command*)

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_WriteCmdDataMastermode or DSPI_HAL_WriteCmdDataMastermodeBlocking 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 would 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

<i>baseAddr</i>	Module base address
<i>command</i>	Pointer to command structure

Returns

The command word formatted to the PUSHR data register bit field

9.2.4.41 **uint32_t DSPI_HAL_GetFifoData (uint32_t *baseAddr*, dspi_fifo_t *whichFifo*, uint32_t *whichFifoEntry*)**

Parameters

<i>baseAddr</i>	Module base address
<i>whichFifo</i>	Selects Tx or Rx FIFO, of type dspi_fifo_t.
<i>whichFifoEntry</i>	Selects which FIFO entry to read: 0, 1, 2, or 3.

Returns

The desired FIFO register contents

9.2.4.42 **static void DSPI_HAL_SetHaltInDebugmodeCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]**

Parameters

<i>baseAddr</i>	Module base address
<i>enable</i>	Enables (true) debug mode to halt transfers, else disable to not halt transfer in debug mode.

9.3 DSPI Master Driver

9.3.1 Overview

This section describes the programming interface of the DSPI master mode peripheral driver. The DSPI master mode peripheral driver transfers data to and from external devices on the DSPI bus in master mode. It supports transferring data buffers with a single function call.

9.3.2 DSPI Introduction

The driver is separated into two implementations: interrupt driven and DMA driven. The interrupt driven driver uses interrupts to alert the CPU that the DSPI module needs to service the SPI data transmit and receive operations. The enhanced DMA (eDMA) driven driver uses the eDMA module to transfer data between the buffers located in memory and the DSPI module transmit/receive buffers/FIFOs. Throughout the remainder of this document, the enhanced DMA driven driver will simply be stated as the DMA driven driver. The interrupt driven and DMA driven driver APIs are distinguished by the keyword "edma" in the source file name and by the keyword "Edma" in the API name. Each set of drivers have the same API functionality and are described in the following chapters. Note that the DMA driven driver also uses interrupts to alert the CPU that the DMA has completed its transfer or that one final piece of data still needs to be received which is handled by the IRQ handler in the DMA driven driver. In both the interrupt and DMA drivers, the SPI module interrupts are enabled in the NVIC. In addition, the DMA driven driver requests channels from the eDMA module. Also, this document will refer to either set of drivers simply as the "DSPI master driver" when discussing items that pertain to either driver. Note, when using the DMA driven DSPI driver, you will also need to initialize the eDMA module. An example is shown later under the Initialization chapter.

This is a basic step-by-step process to initialize and transfer SPI data. For API specific examples, see the examples below. The following uses the interrupt driven APIs and a blocking transfer to illustrate a high-level step-by-step usage. The usage of eDMA driver is similar to the interrupt driven driver. Keep in mind that using interrupt and eDMA drivers in the same runtime application is not recommended because you will need to change the SPI interrupt handler. The interrupt driver calls [DSPI_DRV_IRQHandler\(\)](#) and eDMA driver calls [DSPI_DRV_EdmaIRQHandler\(\)](#). See files fsl_dspi_irq.c and fsl_dspi_edma_irq.c for an example of these function calls.

```
// Init the DSPI
DSPI_DRV_MasterInit(masterInstance, &dspiMasterState, &userConfig);
// Configure the SPI bus
DSPI_DRV_MasterConfigureBus(masterInstance, &spiDevice, &calculatedBaudRate);
// optional, normally do not need to adjust delays, but depends on the slave device:
DSPI_DRV_MasterSetDelay(masterInstance, kDspiPcsToSck, delayInNanoSec,
    &calculatedDelay);
// Perform the transfer
DSPI_DRV_MasterTransferBlocking(masterInstance, NULL, s_dspiSourceBuffer,
    s_dspiSinkBuffer, 32, 1000);
// Do other transfers, when done with the DSPI, then de-init to shut it down
DSPI_DRV_MasterDeinit(masterInstance);
```

Note that it is not normally recommended to mix interrupt and DMA driven drivers in the same application. However, should the user decide to do so, they can separately set up and initialize another instance for D-

MA operations. The user can also de-init the current interrupt driven DSPI instance and re-initialize it for DMA operations. Note that since the DMA driven driver also uses interrupts, the user must take care to direct the IRQ handler from the vector table to the desired driver's IRQ handler. Refer to files `fsl_dspi_irq.c` and `fsl_dspi_edma_irq.c` for examples on how to re-direct the IRQ handlers from the vector table to the interrupt-driven and DMA-driven driver IRQ handlers. Such files need to be included in the applications project in order to direct the DSPI interrupt vectors to the proper IRQ handlers. There are also two other files, `fsl_dspi_shared_function.c` and `fsl_dspi_edma_shared_function.c` that direct the interrupts from the vector table to the appropriate master or slave driver interrupt handler by checking the DSPI mode via the HAL function `DSPI_HAL_IsMaster(baseAddr)`. Note that the interrupt driver calls `DSPI_DRV_IRQHandler()` and eDMA driver calls `DSPI_DRV_EdmaIRQHandler()`. See files `fsl_dspi_irq.c` and `fsl_dspi_edma_irq.c` for an example of these function calls.

9.3.3 DSPI Run-time state structures

The DSPI master driver uses a run-time state structure to track the ongoing data transfers. The state structure for the interrupt driven driver is called `dspi_master_state_t` while the state structure for the DMA driven driver is called `dspi_edma_master_state_t`. This structure holds data that the DSPI master peripheral driver uses to communicate between the transfer function and the interrupt handler and other driver functions. The interrupt handler in the interrupt driven also uses this information to keep track of its progress. The user is only required to pass the memory for the run-time state structure. The DSPI master driver populates the members.

9.3.4 DSPI User configuration structures

The DSPI master driver uses instances of the user configuration structure for the DSPI master driver. The user configuration structure for the interrupt driven driver is called `dspi_master_user_config_t` while the user configuration structure for the DMA driven driver is called `dspi_edma_master_user_config_t`. This structure allows the user to configure the most common settings of the DSPI peripheral with a single function call. The user configuration structure is passed into the master driver init function (`DSPI_DRV_MasterInit` or `DSPI_DRV_EdmaMasterInit`). The user configuration structure consists of the following user configuration settings: whichCtar (generally set this to `kDspiCtar0`), option for continuous clock and peripheral chip select, the desired peripheral chip select to use, and the polarity of the peripheral chip select. An example of this is provided later in the document in the Initialization section.

9.3.5 DSPI Device structures

The DSPI master driver uses instances of the `dspi_device_t` or `dspi_edma_device_t` structure to represent the SPI bus configuration required to communicate to an external device that is connected to the bus.

The device structure can be passed into the `DSPI_DRV_MasterConfigureBus` or `DSPI_DRV_EdmaMasterConfigureBus` functions to manually configure the bus for a particular device. For example, if there is only one device connected to the bus, the user might configure it only once. Alternatively the de-

DSPI Master Driver

vice structure can be passed to the data transfer functions where the bus is reconfigured before the transfer is started. The device structure consists of the following settings: bitsPerSec (baud rate in Hz), and the dataBusConfig structure which consists of bits per frame, clock polarity and phase, and data shift direction (msb or lsb).

9.3.6 DSPI Initialization

To initialize the DSPI master driver, call the [DSPI_DRV_MasterInit\(\)](#) or [DSPI_DRV_EdmaMasterInit](#) function and pass the instance number of the DSPI peripheral, the memory allocation for the run-time state structure used by the master driver to keep track of data transfers, and the user configuration ([dsPIC33F_DSPI_MasterUserConfig_t](#) or [dsPIC33F_EDMA_DSPI_MasterUserConfig_t](#)). For the DMA driven driver, the memory allocation for the stcdSrc2CmdDataLast structure will also need to be passed in (note the pointer to this structure needs to be aligned to a 32-byte boundary).

The user first calls the DSPI master initialization to initialize the DSPI module, then calls the DSPI master configuration bus to configure the module for the specific device on the SPI bus. For the interrupt driven case, while the [DSPI_DRV_MasterInit\(\)](#) function initializes the DSPI peripheral, the [DSPI_DRV_MasterConfigureBus\(\)](#) function configures the SPI bus parameters such as bits/frame, clock characteristics, data shift direction, baud rate, and desired chip select. The DMA driven case follows the same logic (except that it uses the EDMA API names) and both examples are provided below. First, the interrupt driven example will be provided followed by the DMA example.

Example code to initialize and configure the DSPI master interrupt driven driver including setting up the user configuration and device structures:

```
// Set up and init the master //
uint32_t masterInstance = 1; // example using DSPI instance 1
dsPIC33F_DSPI_MasterState_t dspiMasterState; // simply allocate memory for this
uint32_t calculatedBaudRate;

// configure the members of the user config //
dsPIC33F_DSPI_MasterUserConfig_t userConfig;
userConfig.isChipSelectContinuous = false;
userConfig.isSckContinuous = false;
userConfig.pcsPolarity = kDspic33F_Pcs_ActiveLow;
userConfig.whichCtar = kDspic33F_Ctar0;
userConfig.whichPcs = kDspic33F_Pcs1;

// init the DSPI module //
DSPI_DRV_MasterInit(masterInstance, &dspiMasterState, &userConfig);

// Define bus configuration.
dsPIC33F_DSPI_Device_t spiDevice;
spiDevice.dataBusConfig.bitsPerFrame = 16;
spiDevice.dataBusConfig.clkPhase = kDspic33F_ClockPhase_FirstEdge;
spiDevice.dataBusConfig.clkPolarity = kDspic33F_ClockPolarity_ActiveHigh
;
spiDevice.dataBusConfig.direction = kDspic33F_MsbFirst;
spiDevice.bitsPerSec = 500000;

// configure the SPI bus //
DSPI_DRV_MasterConfigureBus(masterInstance, &spiDevice, &calculatedBaudRate);
```

Example code to initialize and configure the DSPI master DMA driven driver including setting up the user configuration and device structures:

```

// Declare 32-byte aligned software transfer control descriptor (eDMA requirement)
// This example uses IAR preprocessor pragma syntax. Other methods also include declaring
// a 64-byte region and then aligning the pointer within that region to a 32-byte boundary,
// but this would waste 32-bytes of memory
#pragma data_alignment=32
    edma_software_tcd_t stcdTransferCntTest;

// First, need to init the eDMA peripheral driver.
// NOTE: THIS IS NOT PART OF THE DSPI DRIVER. THIS PART INITIALIZES THE EDMA DRIVER SO
// THAT THE DSPI EDMA DRIVER CAN WORK!
edma_state_t state; // <- The user simply allocates memory for this structure.
edma_user_config_t edmaUserConfig; // <- The user fills out members for this struct.

edmaUserConfig.chnArbitration = kEDMACHnArbitrationRoundrobin;
edmaUserConfig.notHaltOnError = false;

// Actual EDMA initialization
EDMA_DRV_Init(&state, &edmaUserConfig);

// Set up and init the master //
uint32_t masterInstance = 0; // example using DSPI instance 0
dspi_edma_master_state_t dspiMasterState; // simply allocate memory for this
uint32_t calculatedBaudRate;

// configure the members of the user config //
dspi_edma_master_user_config_t userConfig;
userConfig.isChipSelectContinuous = false;
userConfig.isSckContinuous = false;
userConfig.pcsPolarity = kDspiPcs_ActiveLow;
userConfig.whichCtar = kDspiCtar0;
userConfig.whichPcs = kDspiPcs1;

// init the DSPI module //
DSPI_DRV_EdmaMasterInit(masterInstance, &dspiMasterState, &userConfig, &
    stcdTransferCntTest);

// Define bus configuration.
dspi_edma_device_t spiDevice;
spiDevice.dataBusConfig.bitsPerFrame = 16;
spiDevice.dataBusConfig.clkPhase = kDspiClockPhase_FirstEdge;
spiDevice.dataBusConfig.clkPolarity = kDspiClockPolarity_ActiveHigh
    ;
spiDevice.dataBusConfig.direction = kDspiMsbFirst;
spiDevice.bitsPerSec = 500000;

// configure the SPI bus //
DSPI_DRV_EdmaMasterConfigureBus(masterInstance, &spiDevice, &
    calculatedBaudRate);

```

The DSPI also offers an optional API to configure various bus timing delays. This function involves the DSPI module's delay options to "fine tune" some of the signal timings and match the timing needs of a slower peripheral device. This is an optional function that can be called after the DSPI module has been initialized for master mode. The bus timing delays that can be adjusted are listed below:

PCS to SCK Delay: Adjustable delay option between the assertion of the PCS signal to the first SCK edge.

After SCK Delay: Adjustable delay option between the last edge of SCK to the de-assertion of the PCS signal.

Delay after Transfer: Adjustable delay option between the de-assertion of the PCS signal for a frame to the assertion of the PCS signal for the next frame. Note that this is not adjustable for continuous clock mode because this delay is fixed at one SCK period.

DSPI Master Driver

This function takes in as a parameter the desired delay type and the delay value (in nanoseconds), calculates the values needed for the prescaler and scaler. Returning the actual 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. In addition, the function returns an out-of-range status.

This is an example that shows how to use the set the bus timings delay function:

```
uint32_t delayInNanoSec = 200; // example, 200ns
uint32_t calculatedDelay;

// Interrupt example
// kDspiPcsToSck: PCS to SCK Delay option, delayInNanoSec is the user's passed in delay in ns
DSPI_DRV_MasterSetDelay(masterInstance, kDspiPcsToSck, delayInNanoSec,
    &calculatedDelay);

// EDMA example
// kDspiPcsToSck: PCS to SCK Delay option, delayInNanoSec is the user's passed in delay in ns
DSPI_DRV_EdmaMasterSetDelay(masterInstance,
    kDspiPcsToSck, delayInNanoSec, &calculatedDelay);
```

9.3.7 DSPI Transfers

The driver supports two different modes for transferring data: blocking and non-blocking (async). The blocking transfer function waits until the transfer is complete before returning. A timeout parameter is passed into the blocking function. A non-blocking (async) function returns immediately after starting the transfer. It is the responsibility of the user to get the transfer status during the transfer to ascertain when the transfer is complete. As such, additional functions are provided to aid in non-blocking transfers: get transfer status and abort transfer.

Blocking transfer function APIs (interrupt and DMA driven):

```
dspi_status_t DSPI_DRV_MasterTransferBlocking(uint32_t instance
    ,
        const dspi_device_t * restrict device,
        const uint8_t * sendBuffer,
        uint8_t * receiveBuffer,
        size_t transferByteCount,
        uint32_t timeout);

dspi_status_t DSPI_DRV_EdmaMasterTransferBlocking(uint32_t
    instance,
        const dspi_edma_device_t * restrict
    device,
        const uint8_t * sendBuffer,
        uint8_t * receiveBuffer,
        size_t transferByteCount,
        uint32_t timeout);
```

Non-blocking function APIs and associated functions (interrupt and DMA driven):

```
// interrupt-driven transfer function
dspi_status_t DSPI_DRV_MasterTransfer(uint32_t instance,
    const dspi_device_t * restrict device,
```

```

        const uint8_t * sendBuffer,
        uint8_t * receiveBuffer,
        size_t transferByteCount);

// Returns whether the previous transfer is completed (interrupt-driven)
dspi_status_t DSPI_DRV_MasterGetTransferStatus(uint32_t
    instance, uint32_t * framesTransferred);

// Terminates an asynchronous transfer early (interrupt-driven)
dspi_status_t DSPI_DRV_MasterAbortTransfer(uint32_t instance);

// DMA-driven transfer function
dspi_status_t DSPI_DRV_EdmaMasterTransfer(uint32_t instance,
    const dsPIC33F_Edma_Device_t * restrict device,
    const uint8_t * sendBuffer,
    uint8_t * receiveBuffer,
    size_t transferByteCount);

// Returns whether the previous transfer is completed (DMA-driven)
dspi_status_t DSPI_DRV_EdmaMasterGetTransferStatus(
    uint32_t instance, uint32_t * framesTransferred);

// Terminates an asynchronous transfer early (DMA-driven)
dspi_status_t DSPI_DRV_EdmaMasterAbortTransfer(uint32_t
    instance);

```

Example of a blocking transfer (interrupt and DMA driven). Note, first need to initialize the peripheral driver. Refer to the Initialization chapter to perform this first before transferring.

```

// Example blocking transfer function call (interrupt)
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the DSPI_DRV_MasterConfigureBus was sufficient, so we'll pass in NULL for the device
// structure. Also, this example shows that we're transferring 32 bytes with a timeout of 1000us.
DSPI_DRV_MasterTransferBlocking(masterInstance, NULL, s_dspiSourceBuffer,
    s_dspiSinkBuffer, 32, 1000);

// Example blocking transfer function call (DMA)
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the DSPI_DRV_EdmaMasterConfigureBus was sufficient, so we'll pass in NULL for the device
// structure. Also, this example shows that we're transferring 32 bytes with a timeout of 1000us.
DSPI_DRV_EdmaMasterTransferBlocking(masterInstance, NULL,
    s_dspiSourceBuffer,
    s_dspiSinkBuffer, 32, 1000);

```

Example of a non-blocking transfer (interrupt and DMA driven). Note, first need to initialize the peripheral driver. Refer to the Initialization chapter to perform this first before transferring:

```

// Interrupt Example
uint32_t framesXfer;
// Example non-blocking transfer function call
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the DSPI_DRV_MasterConfigureBus was sufficient, so we'll pass in NULL for the device
// structure. Also, this example shows that we're transferring 32 bytes.
DSPI_DRV_MasterTransfer(masterInstance, NULL, s_dspiSourceBuffer, s_dspiSinkBuffer,
    32);

// For non-blocking/async transfers, need to check back to get transfer status, for example
// Where framesXfer returns the number of frames transferred //
DSPI_DRV_MasterGetTransferStatus(masterInstance, &framesXfer);

// Additionally, if for some reason we need to terminate the on-going transfer:
DSPI_DRV_MasterAbortTransfer(masterInstance);

```

DSPI Master Driver

```
// DMA Example
// Example non-blocking transfer function call
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the DSPI_DRV_EdmaMasterConfigureBus was sufficient, so we'll pass in NULL for the device
// structure. Also, this example shows that we're transferring 32 bytes.
DSPI_DRV_EdmaMasterTransfer(masterInstance, NULL, s_dspiSourceBuffer,
    s_dspiSinkBuffer, 32);

// For non-blocking/async transfers, need to check back to get transfer status, for example
// Where framesXfer returns the number of frames transferred //
DSPI_DRV_EdmaMasterGetTransferStatus(masterInstance, &framesXfer);

// Additionally, if for some reason we need to terminate the on-going transfer:
DSPI_DRV_EdmaMasterAbortTransfer(masterInstance);
```

9.3.8 DSPI De-initialization

To de-initialize and shut down the DSPI module, call this function:

```
// interrupt driven
void DSPI_DRV_MasterDeinit(masterInstance);

// DMA driven
void DSPI_DRV_EdmaMasterDeinit(masterInstance);
```

Data Structures

- struct [dsPIC33F_DSPI_Edma_Device_t](#)
Data structure containing information about a device on the SPI bus with EDMA. [More...](#)
- struct [dsPIC33F_DSPI_Edma_Master_State_t](#)
Runtime state structure for the DSPI master driver with EDMA. [More...](#)
- struct [dsPIC33F_DSPI_Edma_Master_User_Config_t](#)
The user configuration structure for the DSPI master driver with EDMA. [More...](#)
- struct [dsPIC33F_DSPI_Device_t](#)
Data structure containing information about a device on the SPI bus. [More...](#)
- struct [dsPIC33F_DSPI_Master_State_t](#)
Runtime state structure for the DSPI master driver. [More...](#)
- struct [dsPIC33F_DSPI_Master_User_Config_t](#)
The user configuration structure for the DSPI master driver. [More...](#)

Variables

- const uint32_t [g_dspiBaseAddr](#) []
Table of base addresses for DSPI instances.
- const IRQn_Type [g_dspiIrqId](#) [HW_SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.
- const uint32_t [g_dspiBaseAddr](#) []
Table of base addresses for DSPI instances.
- const IRQn_Type [g_dspiIrqId](#) [HW_SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.

Initialization and shutdown

- `dspi_status_t DSPI_DRV_EdmaMasterInit (uint32_t instance, dspi_edma_master_state_t *dspiEdmaState, const dspi_edma_master_user_config_t *userConfig, edma_software_tcd_t *stcdSrc2CmdDataLast)`
Initializes a DSPI instance for master mode operation to work with EDMA.
- `void DSPI_DRV_EdmaMasterDeinit (uint32_t instance)`
Shuts down a DSPI instance with the EDMA support.
- `dspi_status_t DSPI_DRV_EdmaMasterSetDelay (uint32_t instance, dspi_delay_type_t whichDelay, uint32_t delayInNanoSec, uint32_t *calculatedDelay)`
Configures the DSPI master mode bus timing delay options with the EDMA support.

Bus configuration

- `dspi_status_t DSPI_DRV_EdmaMasterConfigureBus (uint32_t instance, const dspi_edma_device_t *device, uint32_t *calculatedBaudRate)`
Configures the DSPI port physical parameters to access a device on the bus with the EDMA support.

Blocking transfers

- `dspi_status_t DSPI_DRV_EdmaMasterTransferBlocking (uint32_t instance, const dspi_edma_device_t *restrict device, const uint8_t *sendBuffer, uint8_t *receiveBuffer, size_t transferByteCount, uint32_t timeout)`
Performs a blocking SPI master mode transfer with the EDMA support.

Non-blocking transfers

- `dspi_status_t DSPI_DRV_EdmaMasterTransfer (uint32_t instance, const dspi_edma_device_t *restrict device, const uint8_t *sendBuffer, uint8_t *receiveBuffer, size_t transferByteCount)`
Performs a non-blocking SPI master mode transfer with the EDMA support.
- `dspi_status_t DSPI_DRV_EdmaMasterGetTransferStatus (uint32_t instance, uint32_t *framesTransferred)`
Returns whether the previous transfer is completed with the EDMA support.
- `dspi_status_t DSPI_DRV_EdmaMasterAbortTransfer (uint32_t instance)`
Terminates an asynchronous transfer early with the EDMA support.

Initialization and shutdown

- `dspi_status_t DSPI_DRV_MasterInit (uint32_t instance, dspi_master_state_t *dspiState, const dspi_master_user_config_t *userConfig)`
Initializes a DSPI instance for master mode operation.
- `void DSPI_DRV_MasterDeinit (uint32_t instance)`
Shuts down a DSPI instance.

DSPI Master Driver

- `dspi_status_t DSPI_DRV_MasterSetDelay (uint32_t instance, dspi_delay_type_t whichDelay, uint32_t delayInNanoSec, uint32_t *calculatedDelay)`
Configures the DSPI master mode bus timing delay options.

Bus configuration

- `dspi_status_t DSPI_DRV_MasterConfigureBus (uint32_t instance, const dspi_device_t *device, uint32_t *calculatedBaudRate)`
Configures the DSPI port physical parameters to access a device on the bus.

Blocking transfers

- `dspi_status_t DSPI_DRV_MasterTransferBlocking (uint32_t instance, const dspi_device_t *restrict device, const uint8_t *sendBuffer, uint8_t *receiveBuffer, size_t transferByteCount, uint32_t timeout)`
Performs a blocking SPI master mode transfer.

Non-blocking transfers

- `dspi_status_t DSPI_DRV_MasterTransfer (uint32_t instance, const dspi_device_t *restrict device, const uint8_t *sendBuffer, uint8_t *receiveBuffer, size_t transferByteCount)`
Performs a non-blocking SPI master mode transfer.
- `dspi_status_t DSPI_DRV_MasterGetTransferStatus (uint32_t instance, uint32_t *framesTransferred)`
Returns whether the previous transfer is completed.
- `dspi_status_t DSPI_DRV_MasterAbortTransfer (uint32_t instance)`
Terminates an asynchronous transfer early.

9.3.9 Data Structure Documentation

9.3.9.1 `struct dspi_edma_device_t`

The user must populate the members to set up the DSPI master with EDMA and properly communicate with the SPI device.

Data Fields

- `uint32_t bitsPerSec`
Baud rate in bits per second.

9.3.9.1.0.15 Field Documentation

9.3.9.1.0.15.1 uint32_t dspi_edma_device_t::bitsPerSec

9.3.9.2 struct dspi_edma_master_state_t

This structure holds data used by the DSPI master Peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. The user passes the memory for this run-time state structure. The DSPI master driver populates the members.

Data Fields

- **dspi_ctar_selection_t whichCtar**
Desired Clock and Transfer Attributes Register (CTAR)
- **uint32_t bitsPerFrame**
Desired number of bits per frame.
- **dspi_which_pcs_config_t whichPcs**
Desired Peripheral Chip Select (pcs)
- **bool isChipSelectContinuous**
Option to enable the continuous assertion of chip select between transfers.
- **uint32_t dsipiSourceClock**
Module source clock.
- **volatile bool isTransferInProgress**
True if there is an active transfer.
- **volatile bool isTransferBlocking**
True if transfer is a blocking transaction.
- **semaphore_t irqSync**
Used to wait for ISR to complete its business.
- **edma_chn_state_t dmaCmdData2Fifo**
Structure definition for the eDMA channel.
- **edma_chn_state_t dmaSrc2CmdData**
Structure definition for the eDMA channel.
- **edma_chn_state_t dmaFifo2Receive**
Structure definition for the eDMA channel.
- **edma_software_tcd_t * stcdSrc2CmdDataLast**
Pointer to SW TCD in memory.
- **bool extraByte**
Flag used for 16-bit transfers with odd byte count.
- **uint8_t *restrict rxBuffer**
The buffer into which received bytes are placed.
- **uint32_t rxTransferByteCnt**
Number of bytes to receive.

DSPI Master Driver

9.3.9.2.0.16 Field Documentation

9.3.9.2.0.16.1 **volatile bool dspi_edma_master_state_t::isTransferInProgress**

9.3.9.2.0.16.2 **volatile bool dspi_edma_master_state_t::isTransferBlocking**

9.3.9.2.0.16.3 **semaphore_t dspi_edma_master_state_t::irqSync**

9.3.9.2.0.16.4 **uint8_t* restrict dspi_edma_master_state_t::rxBuffer**

9.3.9.2.0.16.5 **uint32_t dspi_edma_master_state_t::rxTransferByteCnt**

9.3.9.3 struct dspi_edma_master_user_config_t

Use an instance of this structure with the [DSPI_DRV_EdmaMasterInit\(\)](#) function. This allows the user to configure the most common settings of the DSPI peripheral with a single function call.

Data Fields

- **dspi_ctar_selection_t whichCtar**
Desired Clock and Transfer Attributes Register(CTAR)
- **bool isSckContinuous**
Disable or Enable continuous SCK operation.
- **bool isChipSelectContinuous**
Option to enable the continuous assertion of chip select between transfers.
- **dspi_which_pcs_config_t whichPcs**
Desired Peripheral Chip Select (pcs)
- **dspi_pcs_polarity_config_t pcsPolarity**
Peripheral Chip Select (pcs) polarity setting.

9.3.9.3.0.17 Field Documentation

9.3.9.3.0.17.1 **dspi_pcs_polarity_config_t dspi_edma_master_user_config_t::pcsPolarity**

9.3.9.4 struct dspi_device_t

The user must populate these members to set up the DSPI master and properly communicate with the SPI device.

Data Fields

- **uint32_t bitsPerSec**
Baud rate in bits per second.

9.3.9.4.0.18 Field Documentation

9.3.9.4.0.18.1 uint32_t dspi_device_t::bitsPerSec

9.3.9.5 struct dspi_master_state_t

This structure holds data that is used by the DSPI master peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. The user must pass the memory for this run-time state structure. The DSPI master driver populates the members.

Data Fields

- **dspi_ctar_selection_t whichCtar**
Desired Clock and Transfer Attributes Register (CTAR)
- **uint32_t bitsPerFrame**
Desired number of bits per frame.
- **dspi_which_pcs_config_t whichPcs**
Desired Peripheral Chip Select (pcs)
- **bool isChipSelectContinuous**
Option to enable the continuous assertion of chip select between transfers.
- **uint32_t dspiSourceClock**
Module source clock.
- **volatile bool isTransferInProgress**
True if there is an active transfer.
- **const uint8_t *restrict sendBuffer**
The buffer from which transmitted bytes are taken.
- **uint8_t *restrict receiveBuffer**
The buffer into which received bytes are placed.
- **volatile size_t remainingSendByteCount**
Number of bytes remaining to send.
- **volatile size_t remainingReceiveByteCount**
Number of bytes remaining to receive.
- **volatile bool isTransferBlocking**
True if transfer is a blocking transaction.
- **semaphore_t irqSync**
Used to wait for ISR to complete its business.
- **bool extraByte**
Flag used for 16-bit transfers with odd byte count.

DSPI Master Driver

9.3.9.5.0.19 Field Documentation

9.3.9.5.0.19.1 `volatile bool dspi_master_state_t::isTransferInProgress`

9.3.9.5.0.19.2 `const uint8_t* restrict dspi_master_state_t::sendBuffer`

9.3.9.5.0.19.3 `uint8_t* restrict dspi_master_state_t::receiveBuffer`

9.3.9.5.0.19.4 `volatile size_t dspi_master_state_t::remainingSendByteCount`

9.3.9.5.0.19.5 `volatile size_t dspi_master_state_t::remainingReceiveByteCount`

9.3.9.5.0.19.6 `volatile bool dspi_master_state_t::isTransferBlocking`

9.3.9.5.0.19.7 `semaphore_t dspi_master_state_t::irqSync`

9.3.9.6 `struct dspi_master_user_config_t`

Use an instance of this structure with the `DSPI_DRV_MasterInit()` function. This allows the user to configure the most common settings of the DSPI peripheral with a single function call.

Data Fields

- `dspi_ctar_selection_t whichCtar`
Desired Clock and Transfer Attributes Register(CTAR)
- `bool isSckContinuous`
Disable or Enable continuous SCK operation.
- `bool isChipSelectContinuous`
Option to enable the continuous assertion of chip select between transfers.
- `dspi_which_pcs_config_t whichPcs`
Desired Peripheral Chip Select (pcs)
- `dspi_pcs_polarity_config_t pcsPolarity`
Peripheral Chip Select (pcs) polarity setting.

9.3.9.6.0.20 Field Documentation

9.3.9.6.0.20.1 `dspi_pcs_polarity_config_t dspi_master_user_config_t::pcsPolarity`

9.3.10 Function Documentation

9.3.10.1 `dspi_status_t DSPI_DRV_EdmaMasterInit (uint32_t instance, dspi_edma_master_state_t * dsipiEdmaState, const dspi_edma_master_user_config_t * userConfig, edma_software_tcd_t * stcdSrc2CmdDataLast)`

This function uses a DMA-driven method for transferring data. This function initializes the run-time state structure to track the ongoing transfers, ungates the clock to the DSPI module, resets the DSPI module, initializes the module to user defined settings and default settings, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the DSPI module. The CTAR parameter is

special in that it allows the user to have different SPI devices connected to the same DSPI module instance in addition to different peripheral chip selects. Each CTAR contains the bus attributes associated with that particular SPI device. For most use cases, where only one SPI device is connected per DSPI module instance, use CTAR0. This is an example to set up and call the DSPI_DRV_EdmaMasterInit function by passing in these parameters:

```
dspi_edma_master_state_t dspiEdmaState; <- the user allocates memory for this
    structure
uint32_t calculatedBaudRate;
dspi_edma_master_user_config_t userConfig; <- the user populates members for
    this structure
userConfig.isChipSelectContinuous = false;
userConfig.isSckContinuous = false;
userConfig.pcsPolarity = kDspiPcs_ActiveLow;
userConfig.whichCtar = kDspiCtar0;
userConfig.whichPcs = kDspiPcs0;
DSPI_DRV_EdmaMasterInit(masterInstance, &dspiEdmaState, &userConfig);
```

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>dsPIC33F_EdmaMasterState</i>	The pointer to the DSPI EDMA master driver state structure. The user passes the memory for the run-time state structure. The DSPI master driver populates the members. The run-time state structure keeps track of the transfer in progress.
<i>userConfig</i>	The <code>dsPIC33F_EdmaMasterUserConfig</code> user configuration structure. The user populates the members of this structure and passes the pointer of the structure into the function.
<i>stcdSrc2CmdDataLast</i>	This is a pointer to a structure of the <code>stcdSrc2CmdDataLast</code> type. It needs to be aligned to a 32-byte boundary. Some compilers allow you to use a #pragma directive to align a variable to a desired boundary.

Returns

An error code or `kStatus_DSPI_Success`.

9.3.10.2 void DSPI_DRV_EdmaMasterDeinit (uint32_t *instance*)

This function resets the DSPI peripheral, gates its clock, disables any used interrupts to the core, and releases any used DMA channels.

Parameters

DSPI Master Driver

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

9.3.10.3 **dspi_status_t DSPI_DRV_EdmaMasterSetDelay (uint32_t *instance*, dspi_delay_type_t *whichDelay*, uint32_t *delayInNanoSec*, uint32_t * *calculatedDelay*)**

This function involves using the DSPI module delay options to "fine tune" some of the signal timings and match the timing needs of a slower peripheral device. This is an optional function that can be called after the DSPI module has been initialized for master mode. The bus timing delays that can be adjusted are listed below:

PCS to SCK Delay: Adjustable delay option between the assertion of the PCS signal to the first SCK edge.

After SCK Delay: Adjustable delay option between the last edge of SCK to the de-assertion of the PCS signal.

Delay after Transfer: Adjustable delay option between the de-assertion of the PCS signal for a frame to the assertion of the PCS signal for the next frame. Note that this is not adjustable for continuous clock mode because this delay is fixed at one SCK period.

Each of the above delay parameters use both a pre-scalar and scalar value to calculate the needed delay. This function takes in as a parameter the desired delay type and the delay value (in nanoseconds), calculates the values needed for the prescaler and scaler. Returning the actual 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. In addition, the function returns an out-of-range status.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>whichDelay</i>	The desired delay to configure, must be of type <code>dspi_delay_type_t</code>
<i>delayInNano-Sec</i>	The desired delay value in nano-seconds.
<i>calculated-Delay</i>	The calculated delay that best matches the desired delay (in nano-seconds).

Returns

Either kStatus_DSPI_Success or kStatus_DSPI_OutOfRange if the desired delay exceeds the capability of the device.

9.3.10.4 `dspi_status_t DSPI_DRV_EdmaMasterConfigureBus (uint32_t instance, const dspi_edma_device_t * device, uint32_t * calculatedBaudRate)`

The term "device" is used to indicate the SPI device for which the DSPI master is communicating. The user has two options to configure the device parameters: pass in the pointer to the device configuration structure to the desired transfer function (see DSPI_DRV_EdmaMasterTransferBlocking or DSPI_DRV_EdmaMasterTransfer) or pass it in to the DSPI_DRV_EdmaMasterConfigureBus function. The user can pass in a device structure to the transfer function which contains the parameters for the bus (the transfer function then calls this function). However, the user has the option to call this function directly especially to get the calculated baud rate, at which point they may pass in NULL for the device structure in the transfer function (assuming they have called this configure bus function first). This is an example to set up the `dspi_device_t` structure to call the DSPI_DRV_EdmaMasterConfigureBus function by passing in these parameters:

```
dspi_edma_device_t spiDevice;
spiDevice.dataBusConfig.bitsPerFrame = 16;
spiDevice.dataBusConfig.clkPhase = kDspiClockPhase_FirstEdge;
spiDevice.dataBusConfig.clkPolarity = kDspiClockPolarity_ActiveHigh
    ;
spiDevice.dataBusConfig.direction = kDspiMsbFirst;
spiDevice.bitsPerSec = 50000;
DSPI_DRV_EdmaMasterConfigureBus(instance, &spiDevice, &calculatedBaudRate);
```

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>device</i>	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration. The device parameters are the desired baud rate (in bits-per-sec), and the data format field which consists of bits-per-frame, clock polarity and phase, and data shift direction.
<i>calculated-BaudRate</i>	The calculated baud rate passed back to the user to determine if the calculated baud rate is close enough to meet the needs. The baud rate never exceeds the desired baud rate.

Returns

An error code or kStatus_DSPI_Success.

DSPI Master Driver

**9.3.10.5 `dspi_status_t DSPI_DRV_EdmaMasterTransferBlocking (uint32_t instance,
const dspi_edma_device_t *restrict device, const uint8_t * sendBuffer, uint8_t *
receiveBuffer, size_t transferByteCount, uint32_t timeout)`**

This function simultaneously sends and receives data on the SPI bus, because the SPI is naturally a full-duplex bus. The function does not return until the transfer is complete.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>device</i>	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration in this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. The device can be configured separately by calling the DSPI_DRV_EdmaMasterConfigureBus function.
<i>sendBuffer</i>	The pointer to the data buffer of the data to send. You may pass NULL for this parameter and bytes with a value of 0 (zero) is sent.
<i>receiveBuffer</i>	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
<i>transferByteCount</i>	The number of bytes to send and receive.
<i>timeout</i>	A timeout for the transfer in microseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a kStatus_SPI_Timeout error returned.

Returns

kStatus_DSPI_Success The transfer was successful, or kStatus_DSPI_Busy Cannot perform transfer because a transfer is already in progress, or kStatus_DSPI_Timeout The transfer timed out and was aborted.

9.3.10.6 **dspi_status_t DSPI_DRV_EdmaMasterTransfer (uint32_t *instance*, const dsPIC33F_Edmamaster_t *restrict *device*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, size_t *transferByteCount*)**

This function returns immediately. The user must check back whether the transfer is complete (using the DSPI_DRV_EdmaMasterGetTransferStatus function). This function simultaneously sends and receives data on the SPI bus because the SPI is a full-duplex bus.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>device</i>	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration in this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. The device can be configured separately by calling the DSPI_DRV_EdmaMasterConfigureBus function.

DSPI Master Driver

<i>sendBuffer</i>	The pointer to the data buffer of the data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) are sent.
<i>receiveBuffer</i>	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
<i>transferByte-Count</i>	The number of bytes to send and receive.

Returns

kStatus_DSPI_Success The transfer was successful, or kStatus_DSPI_Busy Cannot perform transfer because a transfer is already in progress.

9.3.10.7 **dspi_status_t DSPI_DRV_EdmaMasterGetTransferStatus (uint32_t *instance*, uint32_t * *framesTransferred*)**

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>frames-Transferred</i>	Pointer to value populated with the number of frames that have been sent during the active transfer. A frame is defined as the number of bits per frame.

Returns

kStatus_DSPI_Success The transfer has completed successfully, or kStatus_DSPI_Busy The transfer is still in progress. *framesTransferred* is filled with the number of words that have been transferred so far.

9.3.10.8 **dspi_status_t DSPI_DRV_EdmaMasterAbortTransfer (uint32_t *instance*)**

During an a-sync transfer, the user has the option to terminate the transfer early if the transfer is still in progress.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

Returns

kStatus_DSPI_Success The transfer was successful, or kStatus_DSPI_NoTransferInProgress No transfer is currently in progress.

9.3.10.9 **dspi_status_t DSPI_DRV_MasterInit (uint32_t *instance*, dspi_master_state_t * *dspiState*, const dspi_master_user_config_t * *userConfig*)**

This function uses a CPU interrupt driven method for transferring data. This function initializes the run-time state structure to track the ongoing transfers, ungates the clock to the DSPI module, resets the DSPI module, initializes the module to user defined settings and default settings, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the DSPI module. The CTAR parameter is special in that it allows the user to have different SPI devices connected to the same DSPI module instance in addition to different peripheral device selects. Each CTAR contains the bus attributes associated with that particular SPI device. For most use cases where only one SPI device is connected per DSPI module instance, use CTAR0. This is an example to set up the [dspi_master_state_t](#) and the [dspi_master_user_config_t](#) parameters and to call the DSPI_DRV_MasterInit function by passing in these parameters:

```
dspi_master_state_t dspiMasterState; <- the user allocates memory for this structure
uint32_t calculatedBaudRate;
dspi_master_user_config_t userConfig; <- the user populates members for this
    structure
userConfig.isChipSelectContinuous = false;
userConfig.isSckContinuous = false;
userConfig.pcsPolarity = kDspiPcs_ActiveLow;
userConfig.whichCtar = kDspiCtar0;
userConfig.whichPcs = kDspiPcs0;
DSPI_DRV_MasterInit(masterInstance, &dspiMasterState, &userConfig);
```

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>dspiState</i>	The pointer to the DSPI master driver state structure. The user passes the memory for this run-time state structure. The DSPI master driver populates the members. This run-time state structure keeps track of the transfer in progress.

DSPI Master Driver

<i>userConfig</i>	The dsPIC33F DSPI API Reference <code>dspi_master_user_config_t</code> user configuration structure. The user populates the members of this structure and passes the pointer of this structure to the function.
-------------------	---

Returns

An error code or `kStatus_DSPI_Success`.

9.3.10.10 void DSPI_DRV_MasterDeinit (uint32_t *instance*)

This function resets the DSPI peripheral, gates its clock, and disables the interrupt to the core.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

9.3.10.11 dspi_status_t DSPI_DRV_MasterSetDelay (uint32_t *instance*, dspi_delay_type_t *whichDelay*, uint32_t *delayInNanoSec*, uint32_t * *calculatedDelay*)

This function involves the DSPI module's delay options to "fine tune" some of the signal timings and match the timing needs of a slower peripheral device. This is an optional function that can be called after the DSPI module has been initialized for master mode. The bus timing delays that can be adjusted are listed below:

PCS to SCK Delay: Adjustable delay option between the assertion of the PCS signal to the first SCK edge.

After SCK Delay: Adjustable delay option between the last edge of SCK to the de-assertion of the PCS signal.

Delay after Transfer: Adjustable delay option between the de-assertion of the PCS signal for a frame to the assertion of the PCS signal for the next frame. Note that this is not adjustable for continuous clock mode because this delay is fixed at one SCK period.

Each of the above delay parameters use both a pre-scalar and scalar value to calculate the needed delay. This function takes in as a parameter the desired delay type and the delay value (in nanoseconds), calculates the values needed for the prescaler and scaler. Returning the actual 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. In addition, the function returns an out-of-range status.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>whichDelay</i>	The desired delay to configure, must be of type <code>dspi_delay_type_t</code>
<i>delayInNano-Sec</i>	The desired delay value in nano-seconds.
<i>calculated-Delay</i>	The calculated delay that best matches the desired delay (in nano-seconds).

Returns

Either `kStatus_DSPI_Success` or `kStatus_DSPI_OutOfRange` if the desired delay exceeds the capability of the device.

9.3.10.12 `dspi_status_t DSPI_DRV_MasterConfigureBus (uint32_t instance, const dspi_device_t * device, uint32_t * calculatedBaudRate)`

The term "device" is used to indicate the SPI device for which the DSPI master is communicating. The user has two options to configure the device parameters: either pass in the pointer to the device configuration structure to the desired transfer function (see `DSPI_DRV_MasterTransferBlocking` or `DSPI_DRV_MasterTransfer`) or pass it in to the `DSPI_DRV_MasterConfigureBus` function. The user can pass in a device structure to the transfer function which contains the parameters for the bus (the transfer function then calls this function). However, the user has the option to call this function directly especially to get the calculated baud rate, at which point they may pass in `NULL` for the device structure in the transfer function (assuming they have called this configure bus function first). This is an example to set up the `dspi_device_t` structure to call the `DSPI_DRV_MasterConfigureBus` function by passing in these parameters:

```
dspi_device_t spiDevice;
spiDevice.dataBusConfig.bitsPerFrame = 16;
spiDevice.dataBusConfig.clkPhase = kDspiClockPhase_FirstEdge;
spiDevice.dataBusConfig.clkPolarity = kDspiClockPolarity_ActiveHigh
;
spiDevice.dataBusConfig.direction = kDspiMsbFirst;
spiDevice.bitsPerSec = 50000;
DSPI_DRV_MasterConfigureBus(instance, &spiDevice, &calculatedBaudRate);
```

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

DSPI Master Driver

<i>device</i>	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration. The device parameters are the desired baud rate (in bits-per-sec), and the data format field which consists of bits-per-frame, clock polarity and phase, and data shift direction.
<i>calculated-BaudRate</i>	The calculated baud rate passed back to the user to determine if the calculated baud rate is close enough to meet the needs. The baud rate never exceeds the desired baud rate.

Returns

An error code or [kStatus_DSPI_Success](#).

9.3.10.13 `dspi_status_t DSPI_DRV_MasterTransferBlocking (uint32_t instance, const dspi_device_t *restrict device, const uint8_t * sendBuffer, uint8_t * receiveBuffer, size_t transferByteCount, uint32_t timeout)`

This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus. The function does not return until the transfer is complete.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>device</i>	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration in this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. The device can be configured separately by calling the DSPI_DRV_MasterConfigureBus function.
<i>sendBuffer</i>	The pointer to the data buffer of the data to send. You may pass NULL for this parameter and bytes with a value of 0 (zero) is sent.
<i>receiveBuffer</i>	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
<i>transferByte-Count</i>	The number of bytes to send and receive.
<i>timeout</i>	A timeout for the transfer in microseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a kStatus_SPI_Timeout error returned.

Returns

[kStatus_DSPI_Success](#) The transfer was successful, or [kStatus_DSPI_Busy](#) Cannot perform transfer because a transfer is already in progress, or [kStatus_DSPI_Timeout](#) The transfer timed out and was aborted.

9.3.10.14 dspi_status_t DSPI_DRV_MasterTransfer (uint32_t *instance*, const dspi_device_t **restrict device*, const uint8_t **sendBuffer*, uint8_t **receiveBuffer*, size_t *transferByteCount*)

This function returns immediately. The user must check back to check whether the transfer is complete (using the DSPI_DRV_MasterGetTransferStatus function). This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>device</i>	Pointer to the device information structure. This structure contains the settings for the SPI bus configuration in this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. The device can be configured separately by calling the DSPI_DRV_MasterConfigureBus function.
<i>sendBuffer</i>	The pointer to the data buffer of the data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) are sent.
<i>receiveBuffer</i>	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
<i>transferByte-Count</i>	The number of bytes to send and receive.

Returns

kStatus_DSPI_Success The transfer was successful, or kStatus_DSPI_Busy Cannot perform transfer because a transfer is already in progress.

9.3.10.15 dspi_status_t DSPI_DRV_MasterGetTransferStatus (uint32_t *instance*, uint32_t **framesTransferred*)

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

DSPI Master Driver

<i>frames-Transferred</i>	Pointer to value that is filled in with the number of frames that have been sent in the active transfer. A frame is defined as the number of bits per frame.
---------------------------	--

Returns

kStatus_DSPI_Success The transfer has completed successfully, or kStatus_DSPI_Busy The transfer is still in progress. framesTransferred is filled with the number of words that have been transferred so far.

9.3.10.16 **dspi_status_t DSPI_DRV_MasterAbortTransfer (uint32_t *instance*)**

During an a-sync transfer, the user has the option to terminate the transfer early if the transfer is still in progress.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

Returns

kStatus_DSPI_Success The transfer was successful, or kStatus_DSPI_NoTransferInProgress No transfer is currently in progress.

9.3.11 Variable Documentation

9.3.11.1 **const uint32_t g_dspiBaseAddr[]**

9.3.11.2 **const IRQn_Type g_dspilrqId[HW_SPI_INSTANCE_COUNT]**

9.3.11.3 **const uint32_t g_dspiBaseAddr[]**

9.3.11.4 **const IRQn_Type g_dspilrqId[HW_SPI_INSTANCE_COUNT]**

9.4 DSPI Slave Driver

9.4.1 Overview

This chapter describes the programming interface of the DSPI slave mode peripheral driver. The DSPI slave peripheral driver supports using the SPI peripheral in slave mode. It supports transferring buffers of data with a single function call. When the DSPI is configured for slave mode operations, it must first be set up to perform a transfer and then wait for the master to initiate the transfer.

The driver is separated into two implementations: interrupt driven and DMA driven. The interrupt driven driver uses interrupts to alert the CPU that the DSPI module needs to service the SPI data transmit and receive operations. The enhanced DMA (eDMA) driven driver uses the eDMA module to transfer data between the buffers located in memory and the DSPI module transmit/receive buffers/FIFOs. Throughout the remainder of this document, the enhanced DMA driven driver will simply be stated as the DMA driven driver. The interrupt driven and DMA driven driver APIs are distinguished by the keyword "edma" in the source file name and by the keyword "Edma" in the API name. Each set of drivers have the same API functionality and are described in the following chapters. Note that the DMA driven driver also uses interrupts to alert the CPU that the DMA has completed its transfer or that one final piece of data still needs to be received which is handled by the IRQ handler in the DMA driven driver. In both the interrupt and DMA drivers, the SPI module interrupts are enabled in the NVIC. In addition, the DMA driven driver requests channels from the eDMA module. Also, this document will refer to either set of drivers simply as the "DSPI slave driver" when discussing items that pertain to either driver. Note, when using the DMA driven DSPI driver, you will also need to initialize the eDMA module. An example is shown later under the Initialization chapter.

The following is a basic step-by-step overview of how to setup the DSPI for SPI slave mode operations. For API specific examples, refer to the examples below. The following uses the interrupt driven APIs and a blocking transfer to illustrate a high-level step-by-step usage. The usage of eDMA driver is similar to interrupt driven driver. Keep in mind that using interrupt and eDMA drivers in the same runtime application is not recommended as you will need to change the SPI interrupt handler. The interrupt driver calls `DSPI_DRV_IRQHandler()` and eDMA driver calls `DSPI_DRV_EdmaIRQHandler()`. Refer to files `fsl_dspi_irq.c` and `fsl_dspi_edma_irq.c` for an example of these function calls.

```
// Init the DSPI
DSPI_DRV_SlaveInit(slaveInstance, &dspiSlaveState, &userConfig);
// Perform the transfer (however, waits for master to initiate the transfer)
DSPI_DRV_SlaveTransferBlocking(slaveInstance, s_dspiSourceBuffer,
    s_dspiSinkBuffer, 32, 1000);
// Do other transfers, when done with the DSPI, then de-init to shut it down
DSPI_DRV_SlaveDeinit(slaveInstance);
```

Note that it is not normally recommended to mix interrupt and DMA driven drivers in the same application. However, should the user decide to do so, they can separately set up and initialize another instance for DMA operations. The user can also de-init the current interrupt driven DSPI instance and re-initialize it for DMA operations. Note that since the DMA driven driver also uses interrupts, the user must take care to direct the IRQ handler from the vector table to the desired driver's IRQ handler. Refer to files `fsl_dspi_irq.c` and `fsl_dspi_edma_irq.c` for examples on how to re-direct the IRQ handlers from the vector table to the interrupt-driven and DMA-driven driver IRQ handlers. Such files need to be included in the applications

DSPI Slave Driver

project in order to direct the DSPI interrupt vectors to the proper IRQ handlers. There are also two other files, `fsl_dspi_shared_function.c` and `fsl_dspi_edma_shared_function.c` that direct the interrupts from the vector table to the appropriate master or slave driver interrupt handler by checking the DSPI mode via the HAL function `DSPI_HAL_IsMaster(baseAddr)`.

9.4.2 DSPI Runtime state of the DSPI slave driver

The DSPI slave driver uses a run-time state structure to track the ongoing data transfers. The state structure for the interrupt driven driver is called `dspi_slave_state_t` while the state structure for the DMA driven driver is called `dspi_edma_slave_state_t`. The structure holds data that the DSPI slave peripheral driver uses to communicate between the transfer function and the interrupt handler and other driver functions. The interrupt handler also uses this information to keep track of its progress. The user is only responsible to pass the memory for this run-time state structure and the DSPI slave driver fills out the members.

9.4.3 DSPI User configuration structures

The DSPI slave driver uses instances of the user configuration structure for the DSPI slave driver. The user configuration structure for the interrupt driven driver is called `dspi_slave_user_config_t` while the user configuration structure for the DMA driven driver is called `dspi_edma_slave_user_config_t`. For this reason, the user can configure the most common settings of the DSPI peripheral with a single function call.

9.4.4 DSPI Setup and Initialization

To initialize the DSPI slave driver, first create and fill in a `dspi_slave_user_config_t` structure for the interrupt driven driver or `dspi_edma_slave_user_config_t` structure for the DMA driven driver. This structure defines the data format settings for the SPI peripheral. The structure is not required after the driver is initialized and can be allocated on the stack. The user also must pass the memory for the run-time state structure.

This is an example code to initialize and configure the driver for interrupt and DMA operations:

```
// declare which module instance you want to use
uint32_t instance = 1;
uint32_t bitCount = 16;

// Interrupt driven
dspi_slave_state_t dspiSlaveState;
// update configs
dspi_slave_user_config_t slaveUserConfig;
slaveUserConfig.dataConfig.clkPhase =
    kDspiClockPhase_FirstEdge;
slaveUserConfig.dataConfig.clkPolarity =
    kDspiClockPolarity_ActiveHigh;
slaveUserConfig.dataConfig.bitsPerFrame = bitCount;
slaveUserConfig.dummyPattern = DSPI_DEFAULT_DUMMY_PATTERN;
```

```

// init the slave (interrupt driven)
DSPI_DRV_SlaveInit(instance, &dsPICSlaveState, &slaveUserConfig);

// DMA driven
// First set up the EDMA peripheral
edma_state_t state; // <- The user simply allocates memory for this structure.
edma_user_config_t edmaUserConfig; // <- The user fills out members for this struct.
edmaUserConfig.chnArbitration = kEDMACHnArbitrationRoundRobin;
EDMA_DRV_Init(&state, &edmaUserConfig);

dsPIC_edma_slave_state_t dsPICEdmaSlaveState;
// update configs
dsPIC_edma_slave_user_config_t slaveEdmaUserConfig;
slaveEdmaUserConfig.dataConfig.clkPhase =
    kDspicClockPhase_FirstEdge;
slaveEdmaUserConfig.dataConfig.clkPolarity =
    kDspicClockPolarity_ActiveHigh;
slaveEdmaUserConfig.dataConfig.bitsPerFrame = bitCount;
slaveEdmaUserConfig.dummyPattern = DSPI_DEFAULT_DUMMY_PATTERN;

// init the slave (DMA driven)
DSPI_DRV_EdmaSlaveInit(instance, &dsPICEdmaSlaveState, &slaveEdmaUserConfig);

```

9.4.5 DSPI Blocking and non-blocking

The DSPI slave driver has two types of transfer functions, blocking and non-blocking call. With non-blocking calls, the user starts the transfer and then waits for event flags to set. kDspiTxDone indicates the transmission is done and kDspiRxDone indicates reception is done. With the blocking call, the function only returns after the related process is all done.

Here is an example of blocking and non-blocking call (interrupt driven)

```

dsPIC_status_t result;
// Blocking call example
result = DSPI_DRV_SlaveTransferBlocking(instance, // number of SPI peripheral
                                         sendBuffer, // pointer to transmit buffer, can be NULL
                                         receiveBuffer, // pointer to receive buffer, can be NULL
                                         transferSize, // size of receive and receive data
                                         10000); // Time out after 10000ms
// Check the result to know if the transfer was successful.

// Non-blocking call example
result = DSPI_DRV_SlaveTransfer(instance, // number of SPI peripheral
                                 sendBuffer, // pointer to transmit buffer, can be NULL
                                 receiveBuffer, // pointer to receive buffer, can be NULL
                                 transferSize); // size of receive and receive data

// Wait for transfer done
while(kStatus_DSPI_Success != DSPI_DRV_SlaveGetTransferStatus(instance, NULL
));
// Must check the value of osaStatus to know if the transfer was successful.

// User has the option to terminate a transfer in progress
DSPI_DRV_SlaveAbortTransfer(instance);

```

Additionally, the DSPI supports eDMA transfers. To use the DSPI with DMA, see the following example:

```
dsPIC_status_t result;
```

DSPI Slave Driver

```
// Blocking call example
result = DSPI_DRV_EdmaSlaveTransferBlocking(instance, // number of SPI
                                             peripheral
                                             sendBuffer,    // pointer to transmit buffer, can be NULL
                                             receiveBuffer, // pointer to receive buffer, can be NULL
                                             transferSize,  // size of receive and receive data
                                             10000);        // Time out after 10000ms
// Check the result to know if the transfer was successful.

// Non-blocking call example
result = DSPI_DRV_EdmaSlaveTransfer(instance, // number of SPI peripheral
                                     sendBuffer,      // pointer to transmit buffer, can be NULL
                                     receiveBuffer,   // pointer to receive buffer, can be NULL
                                     transferSize);  // size of receive and receive data

// Wait for transfer done
while(kStatus_DSPI_Success != DSPI_DRV_EdmaSlaveGetTransferStatus(
    instance, NULL));
// Must check the value of osaStatus to know if the transfer was successful.

// User has the option to terminate a transfer in progress
DSPI_DRV_EdmaSlaveAbortTransfer(instance);
```

9.4.6 DSPI De-initialization

To de-initialize and shut down the DSPI module, call the function:

```
void DSPI_DRV_SlaveDeinit(instance);
```

If using eDMA driven, call the function

```
void DSPI_DRV_EdmaSlaveDeinit(instance);
```

Data Structures

- struct [dsPIC33F_DSPI_EdmaSlaveUserConfig_t](#)
User configuration structure for the DSPI slave driver. [More...](#)
- struct [dsPIC33F_DSPI_EdmaSlaveState_t](#)
Runtime state structure of the DSPI slave driver. [More...](#)
- struct [dsPIC33F_DSPI_SlaveUserConfig_t](#)
User configuration structure for the DSPI slave driver. [More...](#)
- struct [dsPIC33F_DSPI_SlaveState_t](#)
Runtime state of the DSPI slave driver. [More...](#)

Macros

- #define [DSPI_EDMA_DEFAULT_DUMMY_PATTERN](#) (0u)
Dummy pattern, that DSPI slave will send when transmit data was not configured.
- #define [DSPI_DEFAULT_DUMMY_PATTERN](#) (0x0U)
Dummy pattern, that DSPI slave will send when transmit data was not configured.

Variables

- `dspi_data_format_config_t dspi_edma_slave_user_config_t::dataConfig`
Data format configuration structure.
- `uint16_t dspi_edma_slave_user_config_t::dummyPattern`
Dummy data value.
- `uint32_t dspi_edma_slave_state_t::bitsPerFrame`
Desired number of bits per frame.
- `dspi_status_t dspi_edma_slave_state_t::status`
Current state of slave.
- `event_t dspi_edma_slave_state_t::event`
Event to notify waiting task.
- `uint16_t dspi_edma_slave_state_t::errorCount`
Driver error count.
- `uint32_t dspi_edma_slave_state_t::dummyPattern`
Dummy data will be send when do not have data in transmit buffer.
- `volatile bool dspi_edma_slave_state_t::isTransferInProgress`
True if there is an active transfer.
- `const uint8_t *restrict dspi_edma_slave_state_t::sendBuffer`
Pointer to transmit buffer.
- `uint8_t *restrict dspi_edma_slave_state_t::receiveBuffer`
Pointer to receive buffer.
- `volatile int32_t dspi_edma_slave_state_t::remainingSendByteCount`
Number of bytes remaining to send.
- `volatile int32_t dspi_edma_slave_state_t::remainingReceiveByteCount`
Number of bytes remaining to receive.
- `uint8_t dspi_edma_slave_state_t::extraReceiveByte`
Number of extra receive bytes.
- `bool dspi_edma_slave_state_t::isSync`
Indicates the function call is sync or async.
- `edma_chn_state_t dspi_edma_slave_state_t::edmaTxChannel`
Structure definition for the eDMA channel.
- `edma_chn_state_t dspi_edma_slave_state_t::edmaRxChannel`
Structure definition for the eDMA channel.

Initialization and shutdown

- `dspi_status_t DSPI_DRV_EdmaSlaveInit (uint32_t instance, dspi_edma_slave_state_t *dsSpiState, const dspi_edma_slave_user_config_t *slaveConfig)`
Initializes a DSPI instance for a slave mode operation, using eDMA mechanism.
- `void DSPI_DRV_EdmaSlaveDeinit (uint32_t instance)`
Shuts down a DSPI instance - eDMA mechanism.

Blocking transfers

- `dspi_status_t DSPI_DRV_EdmaSlaveTransferBlocking (uint32_t instance, const uint8_t *sendBuffer, uint8_t *receiveBuffer, uint32_t transferByteCount, uint32_t timeOut)`
Transfers data on SPI bus using eDMA and blocking call.

DSPI Slave Driver

Non-blocking transfers

- `dspi_status_t DSPI_DRV_EdmaSlaveTransfer` (uint32_t instance, const uint8_t *sendBuffer, uint8_t *receiveBuffer, uint32_t transferByteCount)
Starts transfer data on SPI bus using eDMA.
- `dspi_status_t DSPI_DRV_EdmaSlaveAbortTransfer` (uint32_t instance)
Abort transfer that started by non-blocking call eDMA transfer function.
- `dspi_status_t DSPI_DRV_EdmaSlaveGetTransferStatus` (uint32_t instance, uint32_t *framesTransferred)
Returns whether the previous transfer is finished.

Initialization and shutdown

- `dspi_status_t DSPI_DRV_SlaveInit` (uint32_t instance, `dspi_slave_state_t` *dspiState, const `dspi_slave_user_config_t` *slaveConfig)
Initializes a DSPI instance for a slave mode operation, using interrupt mechanism.
- `void DSPI_DRV_SlaveDeinit` (uint32_t instance)
Shuts down a DSPI instance - interrupt mechanism.

Blocking transfers

- `dspi_status_t DSPI_DRV_SlaveTransferBlocking` (uint32_t instance, const uint8_t *sendBuffer, uint8_t *receiveBuffer, uint32_t transferByteCount, uint32_t timeout)
Transfers data on SPI bus using interrupt and blocking call.

Non-blocking transfers

- `dspi_status_t DSPI_DRV_SlaveTransfer` (uint32_t instance, const uint8_t *sendBuffer, uint8_t *receiveBuffer, uint32_t transferByteCount)
Starts transfer data on SPI bus using interrupt and non-blocking call.
- `dspi_status_t DSPI_DRV_SlaveAbortTransfer` (uint32_t instance)
Abort transfer that started by non-blocking call transfer function.
- `dspi_status_t DSPI_DRV_SlaveGetTransferStatus` (uint32_t instance, uint32_t *framesTransferred)
Returns whether the previous transfer is finished.

9.4.7 Data Structure Documentation

9.4.7.1 `struct dspi_edma_slave_user_config_t`

Data Fields

- `dspi_data_format_config_t dataConfig`
Data format configuration structure.
- `uint16_t dummyPattern`

Dummy data value.

9.4.7.2 struct dspi_edma_slave_state_t

This structure holds data that is used by the DSPI slave peripheral driver to communicate between the transfer function and the interrupt handler. The user needs to pass in the memory for this structure and the driver fills out the members.

Data Fields

- `uint32_t bitsPerFrame`
Desired number of bits per frame.
- `dspi_status_t status`
Current state of slave.
- `event_t event`
Event to notify waiting task.
- `uint16_t errorCount`
Driver error count.
- `uint32_t dummyPattern`
Dummy data will be send when do not have data in transmit buffer.
- `volatile bool isTransferInProgress`
True if there is an active transfer.
- `const uint8_t *restrict sendBuffer`
Pointer to transmit buffer.
- `uint8_t *restrict receiveBuffer`
Pointer to receive buffer.
- `volatile int32_t remainingSendByteCount`
Number of bytes remaining to send.
- `volatile int32_t remainingReceiveByteCount`
Number of bytes remaining to receive.
- `uint8_t extraReceiveByte`
Number of extra receive bytes.
- `bool isSync`
Indicates the function call is sync or async.
- `edma_chn_state_t edmaTxChannel`
Structure definition for the eDMA channel.
- `edma_chn_state_t edmaRxChannel`
Structure definition for the eDMA channel.

9.4.7.3 struct dspi_slave_user_config_t

Data Fields

- `dspi_data_format_config_t dataConfig`
Data format configuration structure.
- `uint16_t dummyPattern`
Dummy data value.

9.4.7.4 struct dsPIC33F_DSPI_Drv_EdmaSlaveInit

This structure holds data that is used by the DSPI slave peripheral driver to communicate between the transfer function and the interrupt handler. The user needs to pass in the memory for this structure and the driver fills out the members.

Data Fields

- `uint32_t bitsPerFrame`
Desired number of bits per frame.
- `dsPIC33F_DSPI_Status_t status`
Current state of slave.
- `event_t event`
Event to notify waiting task.
- `uint16_t errorCount`
Driver error count.
- `uint32_t dummyPattern`
Dummy data will be send when do not have data in transmit buffer.
- `volatile bool isTransferInProgress`
True if there is an active transfer.
- `const uint8_t *restrict sendBuffer`
Pointer to transmit buffer.
- `uint8_t *restrict receiveBuffer`
Pointer to receive buffer.
- `volatile int32_t remainingSendByteCount`
Number of bytes remaining to send.
- `volatile int32_t remainingReceiveByteCount`
Number of bytes remaining to receive.
- `bool isSync`
Indicates the function call is sync or async.

9.4.7.4.0.21 Field Documentation

9.4.7.4.0.21.1 volatile bool dsPIC33F_DSPI_Drv_EdmaSlaveInit::isTransferInProgress

9.4.7.4.0.21.2 volatile int32_t dsPIC33F_DSPI_Drv_EdmaSlaveInit::remainingSendByteCount

9.4.7.4.0.21.3 volatile int32_t dsPIC33F_DSPI_Drv_EdmaSlaveInit::remainingReceiveByteCount

9.4.8 Function Documentation

9.4.8.1 dsPIC33F_DSPI_Status_t DSPI_DRV_EdmaSlaveInit (`uint32_t instance`, `dsPIC33F_DSPI_EdmaSlaveState_t * dsPIC33F_DSPI_EdmaSlaveState`, `const dsPIC33F_DSPI_EdmaSlaveUserConfig_t * slaveConfig`)

This function un-gates the clock to the DSPI module, initializes the DSPI for slave mode and initializes e-DMA channels. Once initialized, the DSPI module is configured in slave mode and user can start transmit, receive data by calls send, receive, transfer functions. This function indicates DSPI slave will use eDMA

mechanism.

DSPI Slave Driver

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>dspiState</i>	The pointer to the DSPI slave driver state structure.
<i>slaveConfig</i>	The configuration structure dsPIC33E_DSPI_EdmaSlaveUserConfig_t which configures the data bus format.

Returns

An error code or kStatus_DSPI_Success.

9.4.8.2 void DSPI_DRV_EdmaSlaveDeinit (uint32_t *instance*)

Disable DSPI module, gates its clock, change DSPI slave driver state to NonInit for DSPI slave module which is initialized with eDMA mechanism. After de-initialized, user can re-initialize DSPI slave module with other mechanisms.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

9.4.8.3 dsPIC33E_DSPI_EdmaSlaveTransferBlocking (uint32_t *instance*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, uint32_t *transferByteCount*, uint32_t *timeOut*)

This function check driver status, mechanism and transmit/receive data through SPI bus. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transmit and receive progress will be processed. If only receiveBuffer available, receiving will be processed, else transmitting will be processed. This function only return when its processes were completed. This function uses eDMA mechanism.

Parameters

<i>instance</i>	The instance number of DSPI peripheral
<i>sendBuffer</i>	The pointer to data that user wants to transmit.
<i>receiveBuffer</i>	The pointer to data that user wants to store received data.

<i>transferByte-Count</i>	The number of bytes to send and receive.
<i>timeOut</i>	The maximum number of miliseconds that function will wait before timed out reached.

Returns

kStatus_DSPI_Success if driver starts to send/receive data successfully. kStatus_DSPI_Error if driver is error and needs to clean error. kStatus_DSPI_Busy if driver is receiving/transmitting data and not available. kStatus_DSPI_Timeout if time out reached while transferring is in progress.

9.4.8.4 **dspi_status_t DSPI_DRV_EdmaSlaveTransfer (uint32_t *instance*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, uint32_t *transferByteCount*)**

This function check driver status then set buffer pointers to receive and transmit SPI data. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transfer is done when kDspiTxDone and kDspiRxDone are set. If only receiveBuffer available, transfer is done when kDspiRxDone flag was set, else transfer is done when kDspiTxDone was set. This function uses eDMA mechanism.

Parameters

<i>instance</i>	The instance number of DSPI peripheral
<i>sendBuffer</i>	The pointer to data that user wants to transmit.
<i>receiveBuffer</i>	The pointer to data that user wants to store received data.
<i>transferByte-Count</i>	The number of bytes to send and receive.

Returns

kStatus_DSPI_Success if driver starts to send/receive data successfully. kStatus_DSPI_Error if driver is error and needs to clean error. kStatus_DSPI_Busy if driver is receiving/transmitting data and not available.

9.4.8.5 **dspi_status_t DSPI_DRV_EdmaSlaveAbortTransfer (uint32_t *instance*)**

This function stops the transfer which started by function [DSPI_DRV_EdmaSlaveTransfer\(\)](#).

DSPI Slave Driver

Parameters

<i>instance</i>	The instance number of DSPI peripheral
-----------------	--

Returns

kStatus_DSPI_Success if everything is ok. kStatus_DSPI_InvalidMechanism if the current transaction does not use eDMA mechanism.

9.4.8.6 **dspi_status_t DSPI_DRV_EdmaSlaveGetTransferStatus (uint32_t *instance*, uint32_t * *framesTransferred*)**

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>frames-Transferred</i>	Pointer to value that is filled in with the number of frames that have been sent in the active transfer. A frame is defined as the number of bits per frame.

Returns

kStatus_DSPI_Success The transfer has completed successfully, or kStatus_DSPI_Busy The transfer is still in progress. framesTransferred is filled with the number of words that have been transferred so far.

9.4.8.7 **dspi_status_t DSPI_DRV_SlaveInit (uint32_t *instance*, dspi_slave_state_t * *dspiState*, const dspi_slave_user_config_t * *slaveConfig*)**

This function un-gates the clock to the DSPI module, initializes the DSPI for slave mode. Once initialized, the DSPI module is configured in slave mode and user can start transmit, receive data by calls send, receive, transfer functions. This function indicates DSPI slave will use interrupt mechanism.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>dspiState</i>	The pointer to the DSPI slave driver state structure.
<i>slaveConfig</i>	The configuration structure dsPIC33F DSPI Slave API Reference which configures the data bus format.

Returns

An error code or kStatus_DSPI_Success.

9.4.8.8 void DSPI_DRV_SlaveDeinit (uint32_t *instance*)

Disable DSPI module, gates its clock, change DSPI slave driver state to NonInit for DSPI slave module which is initialized with interrupt mechanism. After de-initialized, user can re-initialize DSPI slave module with other mechanisms.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

9.4.8.9 dspi_status_t DSPI_DRV_SlaveTransferBlocking (uint32_t *instance*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, uint32_t *transferByteCount*, uint32_t *timeout*)

This function check driver status, mechanism and transmit/receive data through SPI bus. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transmit and receive progress will be processed. If only receiveBuffer available, receiving will be processed, else transmitting will be processed. This function only return when its processes were completed. This function uses interrupt mechanism.

Parameters

<i>instance</i>	The instance number of DSPI peripheral
<i>sendBuffer</i>	The pointer to data that user wants to transmit.
<i>receiveBuffer</i>	The pointer to data that user wants to store received data.
<i>transferByteCount</i>	The number of bytes to send and receive.

DSPI Slave Driver

<i>timeout</i>	The maximum number of miliseconds that function will wait before timed out reached.
----------------	---

Returns

kStatus_DSPI_Success if driver starts to send/receive data successfully. kStatus_DSPI_Error if driver is error and needs to clean error. kStatus_DSPI_Busy if driver is receiving/transmitting data and not available. kStatus_DSPI_Timeout if time out reached while transferring is in progress.

9.4.8.10 **dspi_status_t DSPI_DRV_SlaveTransfer (uint32_t *instance*, const uint8_t * *sendBuffer*, uint8_t * *receiveBuffer*, uint32_t *transferByteCount*)**

This function check driver status then set buffer pointers to receive and transmit SPI data. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transfer is done when kDspiTxDone and kDspiRxDone are set. If only receiveBuffer available, transfer is done when kDspiRxDone flag was set, else transfer is done when kDspiTxDone was set. This function uses interrupt mechanism.

Parameters

<i>instance</i>	The instance number of DSPI peripheral
<i>sendBuffer</i>	The pointer to data that user wants to transmit.
<i>receiveBuffer</i>	The pointer to data that user wants to store received data.
<i>transferByte-Count</i>	The number of bytes to send and receive.

Returns

kStatus_DSPI_Success if driver starts to send/receive data successfully. kStatus_DSPI_Error if driver is error and needs to clean error. kStatus_DSPI_Busy if driver is receiving/transmitting data and not available.

9.4.8.11 **dspi_status_t DSPI_DRV_SlaveAbortTransfer (uint32_t *instance*)**

This function stops the transfer which started by function [DSPI_DRV_SlaveTransfer\(\)](#).

Parameters

<i>instance</i>	The instance number of DSPI peripheral
-----------------	--

Returns

kStatus_DSPI_Success if everything is ok. kStatus_DSPI_InvalidMechanism if the current transaction does not use interrupt mechanism.

9.4.8.12 **dspi_status_t DSPI_DRV_SlaveGetTransferStatus (uint32_t *instance*, uint32_t * *framesTransferred*)**

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>frames-Transferred</i>	Pointer to value that is filled in with the number of frames that have been sent in the active transfer. A frame is defined as the number of bits per frame.

Returns

kStatus_DSPI_Success The transfer has completed successfully, or kStatus_DSPI_Busy The transfer is still in progress. framesTransferred is filled with the number of words that have been transferred so far.

9.4.9 Variable Documentation

9.4.9.1 **volatile bool dspi_edma_slave_state_t::isTransferInProgress**

9.4.9.2 **volatile int32_t dspi_edma_slave_state_t::remainingSendByteCount**

9.4.9.3 **volatile int32_t dspi_edma_slave_state_t::remainingReceiveByteCount**

9.5 DSPI Shared IRQ Driver

9.5.1 Overview

This section describes the programming interface of the DSPI shared IRQ driver for master and slave Peripheral drivers.

Functions

- void [**DSPI_DRV_EdmaIRQHandler**](#) (uint32_t instance)
The function DSPI_DRV_EdmaIRQHandler passes IRQ control to either the master or slave driver.
- void [**DSPI_DRV_IRQHandler**](#) (uint32_t instance)
The function DSPI_DRV_IRQHandler passes IRQ control to either the master or slave driver.

9.5.2 Function Documentation

9.5.2.1 void **DSPI_DRV_EdmaIRQHandler** (uint32_t *instance*)

The address of the IRQ handlers are checked to make sure they are non-zero before they are called. If the IRQ handler's address is zero, it means that driver was not present in the link (because the IRQ handlers are marked as weak). This would actually be a program error, because it means the master/slave config for the IRQ was set incorrectly.

9.5.2.2 void **DSPI_DRV_IRQHandler** (uint32_t *instance*)

The address of the IRQ handlers are checked to make sure they are non-zero before they are called. If the IRQ handler's address is zero, it means that driver was not present in the link (because the IRQ handlers are marked as weak). This would actually be a program error, because it means the master/slave config for the IRQ was set incorrectly.

Chapter 10

Enhanced Direct Memory Access (eDMA)

10.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Direct Memory Access (eDMA) block of Kinetis devices.

Modules

- [eDMA HAL driver](#)
- [eDMA Peripheral driver](#)
- [eDMA request](#)

eDMA HAL driver

10.2 eDMA HAL driver

10.2.1 Overview

This section describes the programming interface of the eDMA HAL driver.

Data Structures

- struct [edma_transfer_config_t](#)
eDMA transfer size configuration. [More...](#)
- struct [edma_minorloop_offset_config_t](#)
eDMA TCD Minor loop mapping configuration [More...](#)
- union [edma_error_status_all_t](#)
Error status of the eDMA module. [More...](#)
- struct [edma_software_tcd_t](#)
eDMA TCD [More...](#)

Enumerations

- enum [edma_status_t](#) { ,
 [kStatus_EDMA_InvalidArgument](#) = 1U,
 [kStatus_EDMA_Fail](#) = 2U }
Error code for the eDMA Driver.
- enum [edma_channel_arbitration_t](#) {
 [kEDMAChnArbitrationFixedPriority](#) = 0U,
 [kEDMAChnArbitrationRoundrobin](#) }
eDMA channel arbitration algorithm used for selection among channels.
- enum [edma_channel_priority_t](#)
eDMA channel priority setting
- enum [edma_modulo_t](#)
eDMA modulo configuration
- enum [edma_transfer_size_t](#)
eDMA transfer configuration
- enum [edma_channel_indicator_t](#) { [kEDMAChannel0](#) = 0U }
eDMA channel configuration.
- enum [edma_bandwidth_config_t](#) {
 [kEDMABandwidthStallNone](#) = 0U,
 [kEDMABandwidthStall4Cycle](#) = 2U,
 [kEDMABandwidthStall8Cycle](#) = 3U }
Bandwidth control configuration.

eDMA HAL driver module level operation

- void [EDMA_HAL_Init](#) (uint32_t baseAddr)
Initializes eDMA module to known state.
- void [EDMA_HAL_CancelTransfer](#) (uint32_t baseAddr)

- Cancels the remaining data transfer.
void **EDMA_HAL_ErrorCancelTransfer** (uint32_t baseAddr)
Cancels the remaining data transfer and treats it as an error condition.
- static void **EDMA_HAL_SetHaltCmd** (uint32_t baseAddr, bool halt)
Halts/Un-halts the DMA Operations.
- static void **EDMA_HAL_SetHaltOnErrorCmd** (uint32_t baseAddr, bool haltOnError)
Halts or does not halt the eDMA module when an error occurs.
- static void **EDMA_HAL_SetDebugCmd** (uint32_t baseAddr, bool enable)
Enables/Disables the eDMA DEBUG mode.

eDMA HAL driver channel priority and arbitration configuration.

- static void **EDMA_HAL_SetChannelPreemptMode** (uint32_t baseAddr, uint32_t channel, bool preempt, bool preemption)
Sets the preempt and preemption feature for the eDMA channel.
- static void **EDMA_HAL_SetChannelPriority** (uint32_t baseAddr, uint32_t channel, **edma_channel_priority_t** priority)
Sets the eDMA channel priority.
- static void **EDMA_HAL_SetChannelArbitrationMode** (uint32_t baseAddr, **edma_channel_arbitration_t** channelArbitration)
Sets the channel arbitration algorithm.

eDMA HAL driver configuration and operation.

- static void **EDMA_HAL_SetMinorLoopMappingCmd** (uint32_t baseAddr, bool enable)
Enables/Disables the minor loop mapping.
- static void **EDMA_HAL_SetContinuousLinkCmd** (uint32_t baseAddr, bool continuous)
Enables or disables the continuous transfer mode.
- static uint32_t **EDMA_HAL_GetErrorStatus** (uint32_t baseAddr)
Gets the error status of the eDMA module.
- void **EDMA_HAL_SetErrorIntCmd** (uint32_t baseAddr, bool enable, **edma_channel_indicator_t** channel)
Enables/Disables the error interrupt for channels.
- bool **EDMA_HAL_GetErrorIntCmd** (uint32_t baseAddr, uint32_t channel)
Checks whether the eDMA channel error interrupt is enabled or disabled.
- static uint32_t **EDMA_HAL_GetErrorIntStatusFlag** (uint32_t baseAddr)
Gets the eDMA error interrupt status.
- static void **EDMA_HAL_ClearErrorIntStatusFlag** (uint32_t baseAddr, **edma_channel_indicator_t** channel)
Clears the error interrupt status for the eDMA channel or channels.
- void **EDMA_HAL_SetDmaRequestCmd** (uint32_t baseAddr, **edma_channel_indicator_t** channel, bool enable)
Enables/Disables the DMA request for the channel or all channels.
- static bool **EDMA_HAL_GetDmaRequestCmd** (uint32_t baseAddr, uint32_t channel)
Checks whether the eDMA channel DMA request is enabled or disabled.
- static bool **EDMA_HAL_GetDmaRequestStatusFlag** (uint32_t baseAddr, uint32_t channel)
Gets the eDMA channel DMA request status.

eDMA HAL driver

- static void `EDMA_HAL_ClearDoneStatusFlag` (uint32_t baseAddr, `edma_channel_indicator_t` channel)
Clears the done status for a channel or all channels.
- static void `EDMA_HAL_TriggerChannelStart` (uint32_t baseAddr, `edma_channel_indicator_t` channel)
Triggers the eDMA channel.
- static bool `EDMA_HAL_GetIntStatusFlag` (uint32_t baseAddr, uint32_t channel)
Gets the eDMA channel interrupt request status.
- static uint32_t `EDMA_HAL_GetAllIntStatusFlag` (uint32_t baseAddr)
Gets the eDMA all channel's interrupt request status.
- static void `EDMA_HAL_ClearIntStatusFlag` (uint32_t baseAddr, `edma_channel_indicator_t` channel)
Clears the interrupt status for the eDMA channel or all channels.

eDMA HAL driver hardware TCD configuration functions.

- void `EDMA_HAL_HTCDClearReg` (uint32_t baseAddr, uint32_t channel)
Clears all registers to 0 for the hardware TCD.
- static void `EDMA_HAL_HTCDSrcAddr` (uint32_t baseAddr, uint32_t channel, uint32_t address)
Configures the source address for the hardware TCD.
- static void `EDMA_HAL_HTCDSrcOffset` (uint32_t baseAddr, uint32_t channel, int16_t offset)
Configures the source address signed offset for the hardware TCD.
- void `EDMA_HAL_HTCDSetAttribute` (uint32_t baseAddr, uint32_t channel, `edma_modulo_t` srcModulo, `edma_modulo_t` destModulo, `edma_transfer_size_t` srcTransferSize, `edma_transfer_size_t` destTransferSize)
Configures the transfer attribute for the eDMA channel.
- void `EDMA_HAL_HTCDSetNbytes` (uint32_t baseAddr, uint32_t channel, uint32_t nbytes)
Configures the nbytes for the eDMA channel.
- uint32_t `EDMA_HAL_HTCDGetNbytes` (uint32_t baseAddr, uint32_t channel)
Gets the nbytes configuration data for the hardware TCD.
- void `EDMA_HAL_HTCDSetMinorLoopOffset` (uint32_t baseAddr, uint32_t channel, `edma_minorloop_offset_config_t` *config)
Configures the minor loop offset for the hardware TCD.
- static void `EDMA_HAL_HTCDSrcLastAdjust` (uint32_t baseAddr, uint32_t channel, int32_t size)
Configures the last source address adjustment for the hardware TCD.
- static void `EDMA_HAL_HTCDSrcDestAddr` (uint32_t baseAddr, uint32_t channel, uint32_t address)
Configures the destination address for the hardware TCD.
- static void `EDMA_HAL_HTCDSrcOffset` (uint32_t baseAddr, uint32_t channel, int16_t offset)
Configures the destination address signed offset for the hardware TCD.
- static void `EDMA_HAL_HTCDSrcLastAdjust` (uint32_t baseAddr, uint32_t channel, uint32_t adjust)
Configures the last source address adjustment.
- void `EDMA_HAL_HTCDSrcScatterGatherLink` (uint32_t baseAddr, uint32_t channel, `edma_software_tcd_t` *stcd)
Configures the memory address for the next transfer TCD for the hardware TCD.

- static void [EDMA_HAL_HTCDSethBandwidth](#) (uint32_t baseAddr, uint32_t channel, [edma_bandwidth_config_t](#) bandwidth)

Configures the bandwidth for the hardware TCD.
- static void [EDMA_HAL_HTCDSethChannelMajorLink](#) (uint32_t baseAddr, uint32_t channel, uint32_t majorChannel, bool enable)

Configures the major channel link for the hardware TCD.
- static uint32_t [EDMA_HAL_HTCDGetMajorLinkChannel](#) (uint32_t baseAddr, uint32_t channel)

Gets the major link channel for the hardware TCD.
- static void [EDMA_HAL_HTCDSethScatterGatherCmd](#) (uint32_t baseAddr, uint32_t channel, bool enable)

Enables/Disables the scatter/gather feature for the hardware TCD.
- static bool [EDMA_HAL_HTCDGetScatterGatherCmd](#) (uint32_t baseAddr, uint32_t channel)

Checks whether the scatter/gather feature is enabled for the hardware TCD.
- static void [EDMA_HAL_HTCDSethDisableDmaRequestAfterTCDDoneCmd](#) (uint32_t baseAddr, uint32_t channel, bool disable)

Disables/Enables the DMA request after the major loop completes for the hardware TCD.
- static void [EDMA_HAL_HTCDSethHalfCompleteIntCmd](#) (uint32_t baseAddr, uint32_t channel, bool enable)

Enables/Disables the half complete interrupt for the hardware TCD.
- static void [EDMA_HAL_HTCDSethIntCmd](#) (uint32_t baseAddr, uint32_t channel, bool enable)

Enables/Disables the interrupt after the major loop completes for the hardware TCD.
- static void [EDMA_HAL_HTCDTriggerChannelStart](#) (uint32_t baseAddr, uint32_t channel)

Triggers the start bits for the hardware TCD.
- static bool [EDMA_HAL_HTCDGetChannelActiveStatus](#) (uint32_t baseAddr, uint32_t channel)

Checks whether the channel is running for the hardware TCD.
- void [EDMA_HAL_HTCDSethChannelMinorLink](#) (uint32_t baseAddr, uint32_t channel, uint32_t linkChannel, bool enable)

Sets the channel minor link for the hardware TCD.
- static bool [EDMA_HAL_HTCDGetChannelMinorLinkStatus](#) (uint32_t baseAddr, uint32_t channel)

Return the channel minor link status for the hardware TCD.
- void [EDMA_HAL_HTCDSethMajorCount](#) (uint32_t baseAddr, uint32_t channel, uint32_t count)

Sets the major iteration count according to minor loop channel link setting.
- uint32_t [EDMA_HAL_HTCDGetBeginMajorCount](#) (uint32_t baseAddr, uint32_t channel)

Gets the number of beginning major counts for the hardware TCD.
- uint32_t [EDMA_HAL_HTCDGetCurrentMajorCount](#) (uint32_t baseAddr, uint32_t channel)

Gets the number of current major counts for the hardware TCD.
- uint32_t [EDMA_HAL_HTCDGetFinishedBytes](#) (uint32_t baseAddr, uint32_t channel)

Gets the number of bytes already transferred for the hardware TCD.
- uint32_t [EDMA_HAL_HTCDGetUnfinishedBytes](#) (uint32_t baseAddr, uint32_t channel)

Gets the number of bytes haven't transferred for the hardware TCD.
- static bool [EDMA_HAL_HTCDGetDoneStatusFlag](#) (uint32_t baseAddr, uint32_t channel)

Gets the channel done status.

EDMA HAL driver software TCD configuration functions.

- static void [EDMA_HAL_STCDSetSrcAddr](#) ([edma_software_tcd_t](#) *stcd, uint32_t address)

Configures the source address for the software TCD.
- static void [EDMA_HAL_STCDSetSrcOffset](#) ([edma_software_tcd_t](#) *stcd, int16_t offset)

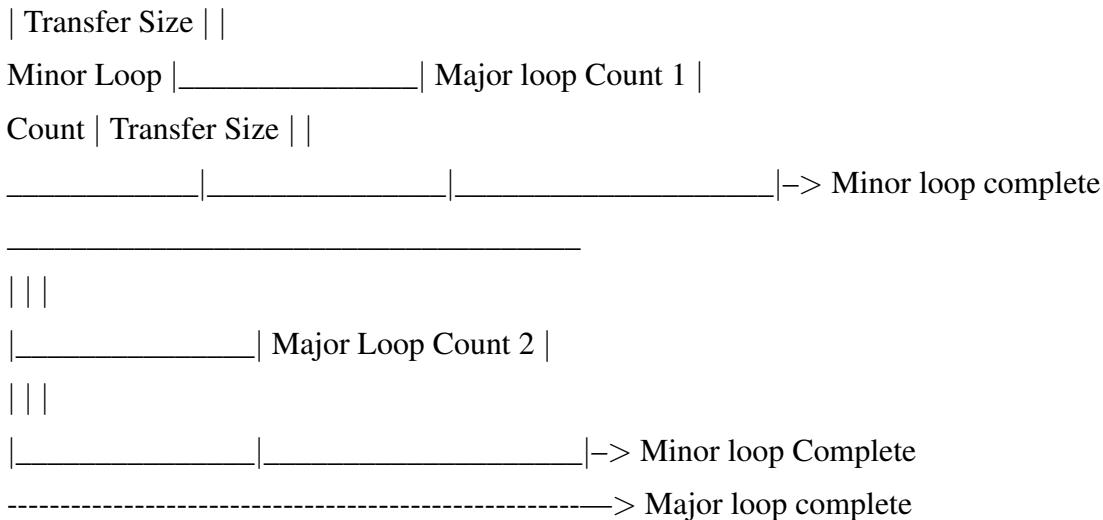
eDMA HAL driver

- Configures the source address signed offset for the software TCD.
 - void **EDMA_HAL_STCDSetAttribute** (*edma_software_tcd_t* *stcd, *edma_modulo_t* srcModulo, *edma_modulo_t* destModulo, *edma_transfer_size_t* srcTransferSize, *edma_transfer_size_t* destTransferSize)
 - Configures the transfer attribute for software TCD.
 - void **EDMA_HAL_STCDSetNbytes** (*uint32_t* baseAddr, *edma_software_tcd_t* *stcd, *uint32_t* nbytes)
 - Configures the nbytes for software TCD.
 - void **EDMA_HAL_STCDSetMinorLoopOffset** (*uint32_t* baseAddr, *edma_software_tcd_t* *stcd, *edma_minorloop_offset_config_t* *config)
 - Configures the minorloop offset for the software TCD.
 - static void **EDMA_HAL_STCDSetSrcLastAdjust** (*edma_software_tcd_t* *stcd, *int32_t* size)
 - Configures the last source address adjustment for the software TCD.
 - static void **EDMA_HAL_STCDSetDestAddr** (*edma_software_tcd_t* *stcd, *uint32_t* address)
 - Configures the destination address for the software TCD.
 - static void **EDMA_HAL_STCDSetDestOffset** (*edma_software_tcd_t* *stcd, *int16_t* offset)
 - Configures the destination address signed offset for the software TCD.
 - static void **EDMA_HAL_STCDSetDestLastAdjust** (*edma_software_tcd_t* *stcd, *uint32_t* adjust)
 - Configures the last source address adjustment.
 - void **EDMA_HAL_STCDSetScatterGatherLink** (*edma_software_tcd_t* *stcd, *edma_software_tcd_t* *nextStcd)
 - Configures the memory address for the next transfer TCD for the software TCD.
 - static void **EDMA_HAL_STCDSetBandwidth** (*edma_software_tcd_t* *stcd, *edma_bandwidth_config_t* bandwidth)
 - Configures the bandwidth for the software TCD.
 - static void **EDMA_HAL_STCDSetChannelMajorLink** (*edma_software_tcd_t* *stcd, *uint32_t* majorChannel, *bool* enable)
 - Configures the major channel link the software TCD.
 - static void **EDMA_HAL_STCDSetScatterGatherCmd** (*edma_software_tcd_t* *stcd, *bool* enable)
 - Enables/Disables the scatter/gather feature for the software TCD.
 - static void **EDMA_HAL_STCDSetDisableDmaRequestAfterTCDDoneCmd** (*edma_software_tcd_t* *stcd, *bool* disable)
 - Enables/Disables the DMA request after the major loop completes for the software TCD.
 - static void **EDMA_HAL_STCDSetHalfCompleteIntCmd** (*edma_software_tcd_t* *stcd, *bool* enable)
 - Enables/Disables the half complete interrupt for the software TCD.
 - static void **EDMA_HAL_STCDSetIntCmd** (*edma_software_tcd_t* *stcd, *bool* enable)
 - Enables/Disables the interrupt after the major loop completes for the software TCD.
 - static void **EDMA_HAL_STCDTriggerChannelStart** (*edma_software_tcd_t* *stcd)
 - Triggers the start bits for the software TCD.
 - void **EDMA_HAL_STCDSetChannelMinorLink** (*edma_software_tcd_t* *stcd, *uint32_t* linkChannel, *bool* enable)
 - Set Channel minor link for software TCD.
 - void **EDMA_HAL_STCDSetMajorCount** (*edma_software_tcd_t* *stcd, *uint32_t* count)
 - Sets the major iteration count according to minor loop channel link setting.
 - void **EDMA_HAL_PushSTCDToHTCD** (*uint32_t* baseAddr, *uint32_t* channel, *edma_software_tcd_t* *stcd)
 - Copy the software TCD configuration to the hardware TCD.
 - *edma_status_t* **EDMA_HAL_STCDSetBasicTransfer** (*uint32_t* baseAddr, *edma_software_tcd_t* *stcd, *edma_transfer_config_t* *config, *bool* enableInt, *bool* disableDmaRequest)
 - Set the basic transfer for software TCD.

10.2.2 Data Structure Documentation

10.2.2.1 struct edma_transfer_config_t

This structure configures the basic source/destination transfer attribute. This figure shows the eDMA's transfer model:



Data Fields

- `uint32_t srcAddr`
Memory address pointing to the source data.
- `uint32_t destAddr`
Memory address pointing to the destination data.
- `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/write is completed.
- `uint32_t srcLastAddrAdjust`
Last source address adjustment.
- `uint32_t destLastAddrAdjust`
Last destination address adjustment.
- `edma_modulo_t srcModulo`
Source address modulo.
- `edma_modulo_t destModulo`
Destination address modulo.
- `uint32_t minorLoopCount`
Minor bytes transfer count.
- `uint16_t majorLoopCount`
Major iteration count.

eDMA HAL driver

10.2.2.1.0.22 Field Documentation

10.2.2.1.0.22.1 uint32_t edma_transfer_config_t::srcAddr

10.2.2.1.0.22.2 uint32_t edma_transfer_config_t::destAddr

10.2.2.1.0.22.3 edma_transfer_size_t edma_transfer_config_t::srcTransferSize

10.2.2.1.0.22.4 edma_transfer_size_t edma_transfer_config_t::destTransferSize

10.2.2.1.0.22.5 int16_t edma_transfer_config_t::srcOffset

10.2.2.1.0.22.6 uint32_t edma_transfer_config_t::srcLastAddrAdjust

10.2.2.1.0.22.7 uint32_t edma_transfer_config_t::destLastAddrAdjust

Note here it is only valid when scatter/gather feature is not enabled.

10.2.2.1.0.22.8 edma_modulo_t edma_transfer_config_t::srcModulo

10.2.2.1.0.22.9 edma_modulo_t edma_transfer_config_t::destModulo

10.2.2.1.0.22.10 uint32_t edma_transfer_config_t::minorLoopCount

Number of bytes to be transferred in each service request of the channel.

10.2.2.1.0.22.11 uint16_t edma_transfer_config_t::majorLoopCount

10.2.2.2 struct edma_minorloop_offset_config_t

Data Fields

- bool **enableSrcMinorloop**
Enable(true) or Disable(false) source minor loop offset.
- bool **enableDestMinorloop**
Enable(true) or Disable(false) destination minor loop offset.
- uint32_t **offset**
Offset for minor loop mapping.

10.2.2.2.0.23 Field Documentation

10.2.2.2.0.23.1 bool edma_minorloop_offset_config_t::enableSrcMinorloop

10.2.2.2.0.23.2 bool edma_minorloop_offset_config_t::enableDestMinorloop

10.2.2.2.0.23.3 uint32_t edma_minorloop_offset_config_t::offset

10.2.2.3 union edma_error_status_all_t

10.2.2.4 struct edma_software_tcd_t

10.2.3 Enumeration Type Documentation

10.2.3.1 enum edma_status_t

Enumerator

kStatus_EDMA_InvalidArgument Parameter is invalid.

kStatus_EDMA_Fail Failed operation.

10.2.3.2 enum edma_channel_arbitration_t

Enumerator

kEDMACHnArbitrationFixedPriority Fixed Priority arbitration is used for selection among channels.

kEDMACHnArbitrationRoundrobin Round-Robin arbitration is used for selection among channels.

10.2.3.3 enum edma_channel_indicator_t

Enumerator

kEDMACHannel0 Channel 0.

10.2.3.4 enum edma_bandwidth_config_t

Enumerator

kEDMABandwidthStallNone No eDMA engine stalls.

kEDMABandwidthStall4Cycle eDMA engine stalls for 4 cycles after each read/write.

kEDMABandwidthStall8Cycle eDMA engine stalls for 8 cycles after each read/write.

10.2.4 Function Documentation

10.2.4.1 void EDMA_HAL_Init(uint32_t *baseAddr*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

10.2.4.2 void EDMA_HAL_CancelTransfer (*uint32_t baseAddr*)

This function stops the executing channel and forces the minor loop to finish. The cancellation takes effect after the last write of the current read/write sequence. The CX clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop had completed.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

10.2.4.3 void EDMA_HAL_ErrorCancelTransfer (*uint32_t baseAddr*)

This function stops the executing channel and forces the minor loop to finish. The cancellation takes effect after the last write of the current read/write sequence. The CX clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop had completed. Additional thing is to treat this operation as an error condition.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

10.2.4.4 static void EDMA_HAL_SetHaltCmd (*uint32_t baseAddr, bool halt*) [inline], [static]

This function stalls/un-stalls the start of any new channels. Executing channels are allowed to be completed.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>halt</i>	Halts (true) or un-halts (false) eDMA transfer.

10.2.4.5 static void EDMA_HAL_SetHaltOnErrorCmd (*uint32_t baseAddr, bool haltOnError*) [inline], [static]

An error causes the HALT bit to be set. Subsequently, all service requests are ignored until the HALT bit is cleared.

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>haltOnError</i>	Halts (true) or not halt (false) eDMA module when an error occurs.

10.2.4.6 static void EDMA_HAL_SetDebugCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function enables/disables the eDMA Debug mode. When in debug mode, the DMA stalls the start of a new channel. Executing channels are allowed to complete. Channel execution resumes either when the system exits debug mode or when the EDBG bit is cleared.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>enable</i>	Enables (true) or Disable (false) eDMA module debug mode.

10.2.4.7 static void EDMA_HAL_SetChannelPreemptMode (*uint32_t baseAddr, uint32_t channel, bool preempt, bool preemption*) [inline], [static]

This function sets the preempt and preemption features.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>preempt</i>	eDMA channel can't suspend a lower priority channel (true). eDMA channel can suspend a lower priority channel (false).
<i>preemption</i>	eDMA channel can be temporarily suspended by the service request of a higher priority channel (true). eDMA channel can't be suspended by a higher priority channel (false).

10.2.4.8 static void EDMA_HAL_SetChannelPriority (*uint32_t baseAddr, uint32_t channel, edma_channel_priority_t priority*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>priority</i>	Priority of the DMA channel. Different channels should have different priority setting inside a group.

10.2.4.9 static void EDMA_HAL_SetChannelArbitrationMode (uint32_t *baseAddr*, edma_channel_arbitration_t *channelArbitration*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel-Arbitration</i>	Round-Robin way for fixed priority way.

10.2.4.10 static void EDMA_HAL_SetMinorLoopMappingCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function enables/disables the minor loop mapping feature. If enabled, the NBYTES is redefined to include the individual enable fields and the NBYTES field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The NBYTES field is reduced when either offset is enabled.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>enable</i>	Enables (true) or Disable (false) minor loop mapping.

10.2.4.11 static void EDMA_HAL_SetContinuousLinkCmd (uint32_t *baseAddr*, bool *continuous*) [inline], [static]

This function enables or disables the continuous transfer. If set, a minor loop channel link does not go through the channel arbitration before being activated again. Upon minor loop completion, the channel activates again if that channel has a minor loop channel link enabled and the link channel is itself.

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>continuous</i>	Enables (true) or Disable (false) continuous transfer mode.

10.2.4.12 static uint32_t EDMA_HAL_GetErrorStatus (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

Returns

Detailed information of the error type in the eDMA module.

10.2.4.13 void EDMA_HAL_SetErrorIntCmd (uint32_t *baseAddr*, bool *enable*, edma_channel_indicator_t *channel*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>enable</i>	Enable(true) or Disable (false) error interrupt.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' error interrupt will be enabled/disabled.

10.2.4.14 bool EDMA_HAL_GetErrorIntCmd (uint32_t *baseAddr*, uint32_t *channel*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

Error interrupt is enabled (true) or disabled (false).

10.2.4.15 static uint32_t EDMA_HAL_GetErrorIntStatusFlag (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

Returns

32 bit variable indicating error channels. If error happens on eDMA channel n, the bit n of this variable is '1'. If not, the bit n of this variable is '0'.

10.2.4.16 static void EDMA_HAL_ClearErrorIntStatusFlag (uint32_t *baseAddr*, edma_channel_indicator_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' error interrupt status will be cleared.

10.2.4.17 void EDMA_HAL_SetDmaRequestCmd (uint32_t *baseAddr*, edma_channel_indicator_t *channel*, bool *enable*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>enable</i>	Enable(true) or Disable (false) DMA request.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels DMA request are enabled/disabled.

10.2.4.18 static bool EDMA_HAL_GetDmaRequestCmd (uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

eDMA HAL driver

<i>channel</i>	eDMA channel number.
----------------	----------------------

Returns

DMA request is enabled (true) or disabled (false).

10.2.4.19 static bool EDMA_HAL_GetDmaRequestStatusFlag (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

Hardware request is triggered in this eDMA channel (true) or not be triggered in this channel (false).

10.2.4.20 static void EDMA_HAL_ClearDoneStatusFlag (*uint32_t baseAddr, edma_channel_indicator_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' done status will be cleared.

10.2.4.21 static void EDMA_HAL_TriggerChannelStart (*uint32_t baseAddr, edma_channel_indicator_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels are tirggere.
----------------	---

10.2.4.22 static bool EDMA_HAL_GetIntStatusFlag (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

Interrupt request happens in this eDMA channel (true) or not happen in this channel (false).

10.2.4.23 static uint32_t EDMA_HAL_GetAllIntStatusFlag (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
-----------------	--

Returns

Interrupt status flag of all channels.

10.2.4.24 static void EDMA_HAL_ClearIntStatusFlag (*uint32_t baseAddr, edma_channel_indicator_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' interrupt status will be cleared.

10.2.4.25 void EDMA_HAL_HTCDClearReg (*uint32_t baseAddr, uint32_t channel*)

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

10.2.4.26 static void EDMA_HAL_HTCDSrcAddr (uint32_t *baseAddr*, uint32_t *channel*, uint32_t *address*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>address</i>	The pointer to the source memory address.

10.2.4.27 static void EDMA_HAL_HTCDSrcOffset (uint32_t *baseAddr*, uint32_t *channel*, int16_t *offset*) [inline], [static]

Sign-extended offset applied to the current source address to form the next-state value as each source read is complete.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>offset</i>	signed-offset for source address.

10.2.4.28 void EDMA_HAL_HTCDSrcAttribute (uint32_t *baseAddr*, uint32_t *channel*, edma_modulo_t *srcModulo*, edma_modulo_t *destModulo*, edma_transfer_size_t *srcTransferSize*, edma_transfer_size_t *destTransferSize*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

<i>srcModulo</i>	enumeration type for an allowed source modulo. The value defines a specific address range specified as the value after the SADDR + SOFF calculation is performed on the original register value. Setting this field provides the ability to implement a circular data. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of the lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with SMOD function restricting the addresses to a 0-modulo-size range.
<i>destModulo</i>	Enum type for an allowed destination modulo.
<i>srcTransferSize</i>	Enum type for source transfer size.
<i>destTransferSize</i>	Enum type for destination transfer size.

10.2.4.29 void EDMA_HAL_HTCDSethNbytes (*uint32_t baseAddr, uint32_t channel, uint32_t nbytes*)

Note here that user need firstly configure the minor loop mapping feature and then call this function.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>nbytes</i>	Number of bytes to be transferred in each service request of the channel

10.2.4.30 *uint32_t EDMA_HAL_HTCDGetNbytes (uint32_t baseAddr, uint32_t channel)*

This function decides whether the minor loop mapping is enabled or whether the source/dest minor loop mapping is enabled. Then, the nbytes are returned accordingly.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

nbytes configuration according to minor loop setting.

eDMA HAL driver

10.2.4.31 void EDMA_HAL_HTCDSetsMinorLoopOffset (uint32_t *baseAddr*, uint32_t *channel*, edma_minorloop_offset_config_t * *config*)

Configures both the enable bits and the offset value. If neither source nor destination offset is enabled, offset is not configured. Note here if source or destination offset is required, the eDMA module EMLM bit will be set in this function. User need to know this side effect.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>config</i>	Configuration data structure for the minor loop offset

10.2.4.32 static void EDMA_HAL_HTCDSetsSrcLastAdjust (uint32_t *baseAddr*, uint32_t *channel*, int32_t *size*) [inline], [static]

Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>size</i>	adjustment value

10.2.4.33 static void EDMA_HAL_HTCDSetsDestAddr (uint32_t *baseAddr*, uint32_t *channel*, uint32_t *address*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>address</i>	The pointer to the destination address.

10.2.4.34 static void EDMA_HAL_HTCDSetsDestOffset (uint32_t *baseAddr*, uint32_t *channel*, int16_t *offset*) [inline], [static]

Sign-extended offset applied to the current source address to form the next-state value as each destination write is complete.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>offset</i>	signed-offset

10.2.4.35 static void EDMA_HAL_HTCDSetsDestLastAdjust (*uint32_t baseAddr, uint32_t channel, uint32_t adjust*) [inline], [static]

This function adds an adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>adjust</i>	adjustment value

10.2.4.36 void EDMA_HAL_HTCDSetsScatterGatherLink (*uint32_t baseAddr, uint32_t channel, edma_software_tcd_t * stcd*)

This function enables the scatter/gather feature for the hardware TCD and configures the next TCD's address. This address points to the beginning of a 0-modulo-32 byte region containing the next transfer T-CD to be loaded into this channel. The channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte. Otherwise, a configuration error is reported.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>stcd</i>	The pointer to the TCD to be linked to this hardware TCD.

10.2.4.37 static void EDMA_HAL_HTCDSetsBandwidth (*uint32_t baseAddr, uint32_t channel, edma_bandwidth_config_t bandwidth*) [inline], [static]

Throttles the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>bandwidth</i>	enum type for bandwidth control

10.2.4.38 static void EDMA_HAL_HTCDSethChannelMajorLink (*uint32_t baseAddr*, *uint32_t channel*, *uint32_t majorChannel*, *bool enable*) [inline], [static]

If the major link is enabled, after the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel start bits.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>majorChannel</i>	channel number for major link
<i>enable</i>	Enables (true) or Disables (false) channel major link.

10.2.4.39 static uint32_t EDMA_HAL_HTCDGetMajorLinkChannel (*uint32_t baseAddr*, *uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

major link channel number

10.2.4.40 static void EDMA_HAL_HTCDSethScatterGatherCmd (*uint32_t baseAddr*, *uint32_t channel*, *bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>enable</i>	Enables (true) /Disables (false) scatter/gather feature.

10.2.4.41 static bool EDMA_HAL_HTCDFGetScatterGatherCmd (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

True stand for enabled. False stands for disabled.

10.2.4.42 static void EDMA_HAL_HTCDFSetDisableDmaRequestAfterTCDDoneCmd (*uint32_t baseAddr, uint32_t channel, bool disable*) [inline], [static]

If disabled, the eDMA hardware automatically clears the corresponding DMA request when the current major iteration count reaches zero.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>disable</i>	Disable (true)/Enable (true) DMA request after TCD complete.

10.2.4.43 static void EDMA_HAL_HTCDFSetHalfCompleteIntCmd (*uint32_t baseAddr, uint32_t channel, bool enable*) [inline], [static]

If set, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This half-way point interrupt request is provided to support the double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's process.

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>enable</i>	Enable (true) /Disable (false) half complete interrupt.

10.2.4.44 static void EDMA_HAL_HTCDS.SetIntCmd (uint32_t *baseAddr*, uint32_t *channel*, bool *enable*) [inline], [static]

If enabled, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches zero.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>enable</i>	Enable (true) /Disable (false) interrupt after TCD done.

10.2.4.45 static void EDMA_HAL_HTCDFTriggerChannelStart (uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

The eDMA hardware automatically clears this flag after the channel begins execution.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

10.2.4.46 static bool EDMA_HAL_HTCDFGetChannelActiveStatus (uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

True stands for running. False stands for not.

10.2.4.47 void EDMA_HAL_HTCDSetsChannelMinorLink (*uint32_t baseAddr, uint32_t channel, uint32_t linkChannel, bool enable*)

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>linkChannel</i>	Channel to be linked on minor loop complete.
<i>enable</i>	Enable (true)/Disable (false) channel minor link.

10.2.4.48 static bool EDMA_HAL_HTCDFGetChannelMinorLinkStatus (uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

Enable (true)/Disable (false) channel minor link.

10.2.4.49 void EDMA_HAL_HTCDFSetMajorCount (uint32_t *baseAddr*, uint32_t *channel*, uint32_t *count*)

Note here that user need to first set the minor loop channel link and then call this function. The execute flow inside this function is dependent on the minor loop channel link setting.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>count</i>	major loop count

10.2.4.50 uint32_t EDMA_HAL_HTCDFGetBeginMajorCount (uint32_t *baseAddr*, uint32_t *channel*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

Begin major counts.

10.2.4.51 **uint32_t EDMA_HAL_HTCDFGetCurrentMajorCount (*uint32_t baseAddr, uint32_t channel*)**

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

Current major counts.

10.2.4.52 **uint32_t EDMA_HAL_HTCDFGetFinishedBytes (*uint32_t baseAddr, uint32_t channel*)**

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

data bytes already transferred

10.2.4.53 **uint32_t EDMA_HAL_HTCDFGetUnfinishedBytes (*uint32_t baseAddr, uint32_t channel*)**

eDMA HAL driver

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

data bytes already transferred

10.2.4.54 static bool EDMA_HAL_HTCDFGetDoneStatusFlag (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

If channel done.

10.2.4.55 static void EDMA_HAL_STCDSetSrcAddr (*edma_software_tcd_t * stcd, uint32_t address*) [inline], [static]

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>address</i>	The source memory address.

10.2.4.56 static void EDMA_HAL_STCDSetSrcOffset (*edma_software_tcd_t * stcd, int16_t offset*) [inline], [static]

Sign-extended offset applied to the current source address to form the next-state value as each source read is complete.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>offset</i>	signed-offset for source address.

**10.2.4.57 void EDMA_HAL_STCDSetAttribute (edma_software_tcd_t *
stcd, edma_modulo_t *srcModulo*, edma_modulo_t *destModulo*,
edma_transfer_size_t *srcTransferSize*, edma_transfer_size_t *destTransferSize*)**

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>srcModulo</i>	enum type for an allowed source modulo. The value defines a specific address range specified as the value after the SADDR + SOFF calculation is performed on the original register value. Setting this field provides the ability to implement a circular data. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of the lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with SMOD function restricting the addresses to a 0-modulo-size range.
<i>destModulo</i>	Enum type for an allowed destination modulo.
<i>srcTransferSize</i>	Enum type for source transfer size.
<i>destTransferSize</i>	Enum type for destination transfer size.

**10.2.4.58 void EDMA_HAL_STCDSetNbytes (uint32_t *baseAddr*, edma_software_tcd_t *
stcd, uint32_t *nbytes*)**

Note here that user need firstly configure the minor loop mapping feature and then call this function.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>stcd</i>	The pointer to the software TCD.
<i>nbytes</i>	Number of bytes to be transferred in each service request of the channel

eDMA HAL driver

**10.2.4.59 void EDMA_HAL_STCDSetMinorLoopOffset (uint32_t *baseAddr*,
edma_software_tcd_t * *stcd*, edma_minorloop_offset_config_t * *config*)**

Configures both the enable bits and the offset value. If neither source nor dest offset is enabled, offset is not configured. Note here if source or destination offset is required, the eDMA module EMLM bit will be set in this function. User need to know this side effect.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>stcd</i>	The pointer to the software TCD.
<i>config</i>	Configuration data structure for the minorloop offset

**10.2.4.60 static void EDMA_HAL_STCDSetSrcLastAdjust (edma_software_tcd_t * *stcd*,
int32_t *size*) [inline], [static]**

Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>size</i>	adjustment value

**10.2.4.61 static void EDMA_HAL_STCDSetDestAddr (edma_software_tcd_t * *stcd*,
uint32_t *address*) [inline], [static]**

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>address</i>	The pointer to the destination addresss.

**10.2.4.62 static void EDMA_HAL_STCDSetDestOffset (edma_software_tcd_t * *stcd*,
int16_t *offset*) [inline], [static]**

Sign-extended offset applied to the current source address to form the next-state value as each destination write is complete.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>offset</i>	signed-offset

10.2.4.63 static void EDMA_HAL_STCDSetDestLastAdjust (*edma_software_tcd_t * stcd*, *uint32_t adjust*) [inline], [static]

This function add an adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>adjust</i>	adjustment value

10.2.4.64 void EDMA_HAL_STCDSetScatterGatherLink (*edma_software_tcd_t * stcd*, *edma_software_tcd_t * nextStcd*)

This function enable the scatter/gather feature for the software TCD and configure the next TCD's address. This address points to the beginning of a 0-modulo-32 byte region containing the next transfer TCD to be loaded into this channel. The channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte. Otherwise, a configuration error is reported.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>nextStcd</i>	The pointer to the TCD to be linked to this software TCD.

10.2.4.65 static void EDMA_HAL_STCDSetBandwidth (*edma_software_tcd_t * stcd*, *edma_bandwidth_config_t bandwidth*) [inline], [static]

Throttles the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch.

eDMA HAL driver

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>bandwidth</i>	enum type for bandwidth control

10.2.4.66 static void EDMA_HAL_STCDSetChannelMajorLink (*edma_software_tcd_t* * *stcd*, *uint32_t* *majorChannel*, *bool enable*) [inline], [static]

If the majorlink is enabled, after the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel start bits.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>majorChannel</i>	channel number for major link
<i>enable</i>	Enables (true) or Disables (false) channel major link.

10.2.4.67 static void EDMA_HAL_STCDSetScatterGatherCmd (*edma_software_tcd_t* * *stcd*, *bool enable*) [inline], [static]

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>enable</i>	Enables (true) /Disables (false) scatter/gather feature.

10.2.4.68 static void EDMA_HAL_STCDSetDisableDmaRequestAfterTCDDoneCmd (*edma_software_tcd_t* * *stcd*, *bool disable*) [inline], [static]

If disabled, the eDMA hardware automatically clears the corresponding DMA request when the current major iteration count reaches zero.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>disable</i>	Disable (true)/Enable (true) dma request after TCD complete.

10.2.4.69 static void EDMA_HAL_STCDSetHalfCompleteIntCmd (edma_software_tcd_t * *stcd*, bool *enable*) [inline], [static]

If set, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This half-way point interrupt request is provided to support the double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's process.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>enable</i>	Enable (true) /Disable (false) half complete interrupt.

10.2.4.70 static void EDMA_HAL_STCDSetIntCmd (edma_software_tcd_t * *stcd*, bool *enable*) [inline], [static]

If enabled, the channel generates an interrupt request by setting the appropriate bit in the interrupt register when the current major iteration count reaches zero.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>enable</i>	Enable (true) /Disable (false) interrupt after TCD done.

10.2.4.71 static void EDMA_HAL_STCDTriggerChannelStart (edma_software_tcd_t * *stcd*) [inline], [static]

The eDMA hardware automatically clears this flag after the channel begins execution.

Parameters

<i>stcd</i>	The pointer to the software TCD.
-------------	----------------------------------

10.2.4.72 void EDMA_HAL_STCDSetChannelMinorLink (edma_software_tcd_t * *stcd*, uint32_t *linkChannel*, bool *enable*)

eDMA HAL driver

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>linkChannel</i>	Channel to be linked on minor loop complete.
<i>enable</i>	Enable (true)/Disable (false) channel minor link.

10.2.4.73 void EDMA_HAL_STCDSetMajorCount (*edma_software_tcd_t * stcd, uint32_t count*)

Note here that user need to first set the minor loop channel link and then call this function. The execute flow inside this function is dependent on the minor loop channel link setting.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>count</i>	major loop count

10.2.4.74 void EDMA_HAL_PushSTCDToHTCD (*uint32_t baseAddr, uint32_t channel, edma_software_tcd_t * stcd*)

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>stcd</i>	The pointer to the software TCD.

10.2.4.75 *edma_status_t* EDMA_HAL_STCDSetBasicTransfer (*uint32_t baseAddr, edma_software_tcd_t * stcd, edma_transfer_config_t * config, bool enableInt, bool disableDmaRequest*)

This function is used to setup the basic transfer for software TCD. The minor loop setting is not involved here cause minor loop's configuration will lay a impact on the global eDMA setting. And the source minor loop offset is relevant to the dest minor loop offset. For these reasons, minor loop offset configuration is treated as an advanced configuration. User can call the [EDMA_HAL_STCDSetMinorLoopOffset\(\)](#) to configure the minor loop offset feature.

Parameters

<i>baseAddr</i>	Register base address for eDMA module.
<i>stcd</i>	The pointer to the software TCD.
<i>config</i>	The pointer to the transfer configuration structure.
<i>enableInt</i>	Enables (true) or Disables (false) interrupt on TCD complete.
<i>disableDma-Request</i>	Disables (true) or Enable (false) dma request on TCD complete.

eDMA Peripheral driver

10.3 eDMA Peripheral driver

10.3.1 Overview

This chapter describes the programming interface of the eDMA Peripheral driver. The eDMA driver requests, configures, and uses eDMA hardware. It supports module initializations and DMA channel configurations.

10.3.2 eDMA Initialization

To initialize the DMA module, call the [EDMA_DRV_Init\(\)](#) function. The user does not need to pass a configuration data structure. This function enables the eDMA module and clock automatically.

10.3.3 eDMA Channel concept

The eDMA module has many channels. Additionally, the EDMA peripheral driver is designed based on a channel concept. All operations should be started by requesting an eDMA channel and ended by freeing an eDMA channel. The user can configure and run operations on the eDMA module by allocating a channel. If a channel is not allocated, a system error may occur.

10.3.4 DMA request concept

A DMA request triggers an eDMA transfer. The DMA request table is available in device configuration chapters in a device reference manual.

10.3.5 eDMA Memory allocation and alignment

The eDMA peripheral driver does not allocate memory dynamically. The user needs to provide the allocated memory pointer for the driver and ensure that the memory is valid. Otherwise, a system error may occur. Prepare three types of memory:

1. The handler memory: [edma_channel_t]. The driver must store the status data for each channel. The edma_channel_t is designed for this purpose.
2. The [edma_software_tcd_t](#). The eDMA supports a TCD chain, which provides either the scatter-gather feature or the loop feature. The eDMA module loads the TCDCs from memory, where TCDs are stored. The user must provide the memory storing software TCDs and the [EDMA_DRV_ConfigScatterGatherTransfer\(\)](#) function or the [EDMA_DRV_ConfigLoopTransfer\(\)](#) function to configure the software TCDs. If those functions fail to configure the software TCDs, use the [EDMA_DRV_PrepDescriptorTransfer\(\)](#) function to configure the TCD. Then, call the [EDMA_DRV_PushDescriptorToReg\(\)](#) function to push the TCD to registers.

3. The status memory. To get the status of the TCD chains, provide the status memory, which the driver fills with chain status.

The start address of the software TCDs must be 32 bytes.

10.3.6 eDMA Call diagram

To use the DMA driver, follow these steps:

1. Initialize the DMA module: [EDMA_DRV_Init\(\)](#).
2. Request a DMA channel: [EDMA_DRV_RequestChannel\(\)](#).
3. Configure the TCD:
 - Configure the TCD chain in a scatter-gather list. UART transmit/receive is the common case.
 - Configure the TCD chain in a loop way. Audio playback/Record is the common case.
 - Configure software TCD and push it to registers. Use the DSPI case to configure and push the TCD to registers.
4. Register callback function: [EDMA_DRV_InstallCallback\(\)](#).
5. Start the DMA channel: [EDMA_DRV_StartChannel\(\)](#).
6. [OPTION] Stop the DMA channel: [EDMA_DRV_StopChannel\(\)](#).
7. Free the DMA channel: [EDMA_DRV_ReleaseChannel\(\)](#).

This is an example code to initialize and configure the driver by configuring the descriptor:

```
EDMA_DRV_Init();

stcd = (edma_software_tcd_t *)(((uint32_t)status + 32) & ~0x1F);

// Prepare the memory space. //
for (i = 0; i < kEdmaTestChainLength; i++)
{
    // Allocate the memory! //
    srcAddr[i] = malloc(kEdmaTestBufferSize);
    destAddr[i] = malloc(kEdmaTestBufferSize);

    // Check whether the allocation is successfully. //
    if (((uint32_t)srcAddr[i] == 0x0U) & ((uint32_t)destAddr[i] == 0x0U))
    {
        printf("Fail to allocate memory for EDMA test! \r\n");
        goto error;
    }

    // Init the memory buffer. //
    for (j = 0; j < kEdmaTestBufferSize; j++)
    {
        srcAddr[i][j] = j;
        destAddr[i][j] = 0;
    }

    srcSG[i].address = (uint32_t)srcAddr[i];
    destSG[i].address = (uint32_t)destAddr[i];
    srcSG[i].length = kEdmaTestBufferSize;
    destSG[i].length = kEdmaTestBufferSize;
}

if (EDMA_DRV_RequestChannel(channel, kDmaRequestMux0AlwaysOn62, &chan_handler) !=
    channel)
{
```

eDMA Peripheral driver

```
printf("Failed to request channel %d !\r\n", channel);
goto error;
}

EDMA_DRV_ConfigScatterGatherTransfer(
    stcd, &chan_handler, kDmaMemoryToMemory,
    0x1U, kEdmaTestWatermarkLevel,
    srcSG, destSG,
    kEdmaTestChainLength);

EDMA_DRV_InstallCallback(&chan_handler, test_callback, &chan_handler);

EDMA_DRV_StartChannel(&chan_handler);
```

For an example to configure the loop mode, see the SAI module driver.

10.3.7 eDMA Extend the driver

The user can call the eDMA HAL driver to extend the application capability.

Data Structures

- struct [edma_user_config_t](#)
The user configuration structure for the eDMA driver. [More...](#)
- struct [edma_chn_state_t](#)
Data structure for the eDMA channel. [More...](#)
- struct [edma_scatter_gather_list_t](#)
Data structure for configuring a discrete memory transfer. [More...](#)
- struct [edma_state_t](#)
Runtime state structure for the eDMA driver. [More...](#)

Macros

- #define [STCD_SIZE](#)(number) ((number + 1) * 32)
Macro for the memory size needed for the software TCD.
- #define [VIRTUAL_CHN_TO_EDMA_MODULE_REGBASE](#)(chn) [g_edmaBaseAddr](#)[chn/FSL_FEATURE_EDMA_MODULE_CHANNEL]
Macro to get the eDMA physical module indicator from the virtual channel indicator.
- #define [VIRTUAL_CHN_TO_EDMA_CHN](#)(chn) (chn%FSL_FEATURE_EDMA_MODULE_CHANNEL)
Macro to get the eDMA physical channel indicator from the virtual channel indicator.
- #define [VIRTUAL_CHN_TO_DMAMUX_MODULE_REGBASE](#)(chn) [g_dmamuxBaseAddr](#)[chn/FSL_FEATURE_DMAMUX_MODULE_CHANNEL]
Macro to get the DMAMUX physical module indicator from the virtual channel indicator.
- #define [VIRTUAL_CHN_TO_DMAMUX_CHN](#)(chn) (chn%FSL_FEATURE_DMAMUX_MODULE_CHANNEL)
Macro to get the DMAMUX physical channel indicator from the virtual channel indicator.

Typedefs

- `typedef void(* edma_callback_t)(void *parameter, edma_chn_status_t status)`
Definition for the eDMA channel callback function.

Enumerations

- `enum edma_chn_status_t {
 kEDMAChnNormal = 0U,
 kEDMAChnIdle,
 kEDMAChnError }`
Channel status for eDMA channel.
- `enum edma_chn_state_type_t {
 kEDMAInvalidChannel = 0xFFU,
 kEDMAAnyChannel = 0xFEU }`
enum type for channel allocation.
- `enum edma_transfer_type_t {
 kEDMAPeripheralToMemory,
 kEDMAMemoryToPeripheral,
 kEDMAMemoryToMemory }`
A type for the DMA transfer.

Variables

- `const uint32_t g_edmaBaseAddr []`
Array for the eDMA module register base address.
- `const uint32_t g_dmamuxBaseAddr []`
Array for DMAMUX module register base address.
- `const IRQn_Type g_edmaIrqId [HW_DMA_INSTANCE_COUNT][FSL_FEATURE_EDMA_MODULE_CHANNEL]`
Two dimensional array for eDMA channel interrupt vector number.
- `const IRQn_Type g_edmaErrIrqId [HW_DMA_INSTANCE_COUNT]`
Array for eDMA module's error interrupt vector number.

eDMA peripheral driver module level functions

- `edma_status_t EDMA_DRV_Init (edma_state_t *edmaState, const edma_user_config_t *userConfig)`
Initializes all eDMA modules in an SOC.
- `edma_status_t EDMA_DRV_Deinit (void)`
Shuts down all eDMA modules.

eDMA Peripheral driver

eDMA peripheral driver channel management functions

- `uint8_t EDMA_DRV_RequestChannel (uint8_t channel, dma_request_source_t source, edma_chn_state_t *chn)`
Requests an eDMA channel dynamically or statically.
- `edma_status_t EDMA_DRV_ReleaseChannel (edma_chn_state_t *chn)`
Releases an eDMA channel.

eDMA peripheral driver transfer setup functions

- `static edma_status_t EDMA_DRV_PrepareDescriptorTransfer (edma_chn_state_t *chn, edma_software_tcd_t *stcd, edma_transfer_config_t *config, bool enableInt, bool disableDmaRequest)`
Sets the descriptor basic transfer for the descriptor.
- `static edma_status_t EDMA_DRV_PrepareDescriptorScatterGather (edma_software_tcd_t *stcd, edma_software_tcd_t *nextStcd)`
Configures the memory address for the next transfer TCD for the software TCD.
- `static edma_status_t EDMA_DRV_PrepareDescriptorChannelLink (edma_software_tcd_t *stcd, uint32_t linkChn)`
Configures the major channel link the software TCD.
- `edma_status_t EDMA_DRV_PushDescriptorToReg (edma_chn_state_t *chn, edma_software_tcd_t *stcd)`
Copies the software TCD configuration to the hardware TCD.
- `edma_status_t EDMA_DRV_ConfigLoopTransfer (edma_chn_state_t *chn, edma_software_tcd_t *stcd, edma_transfer_type_t type, uint32_t srcAddr, uint32_t destAddr, uint32_t size, uint32_t bytesOnEachRequest, uint32_t totalLength, uint8_t number)`
Configures the DMA transfer in a scatter-gather mode.
- `edma_status_t EDMA_DRV_ConfigScatterGatherTransfer (edma_chn_state_t *chn, edma_software_tcd_t *stcd, edma_transfer_type_t type, uint32_t size, uint32_t bytesOnEachRequest, edma_scatter_gather_list_t *srcList, edma_scatter_gather_list_t *destList, uint8_t number)`
Configures the DMA transfer in a scatter-gather mode.

eDMA Peripheral driver channel operation functions

- `edma_status_t EDMA_DRV_StartChannel (edma_chn_state_t *chn)`
Starts an eDMA channel.
- `edma_status_t EDMA_DRV_StopChannel (edma_chn_state_t *chn)`
Stops the eDMA channel.

eDMA Peripheral callback and interrupt functions

- `edma_status_t EDMA_DRV_InstallCallback (edma_chn_state_t *chn, edma_callback_t callback, void *parameter)`
Registers the callback function and the parameter for eDMA channel.
- `void EDMA_DRV_IRQHandler (uint8_t channel)`
IRQ Handler for eDMA channel interrupt.

- void [EDMA_DRV_ErrorIRQHandler](#) (uint8_t instance)
ERROR IRQ Handler for eDMA channel interrupt.

eDMA Peripheral driver miscellaneous functions

- static [edma_chn_status_t EDMA_DRV_GetChannelStatus](#) ([edma_chn_state_t](#) *chn)
Gets the eDMA channel status.
- static uint32_t [EDMA_DRV_GetFinishedBytes](#) ([edma_chn_state_t](#) *chn)
Gets the bytes already transferred for the eDMA channel current TCD.

10.3.8 Data Structure Documentation

10.3.8.1 struct [edma_user_config_t](#)

Use an instance of this structure with the [EDMA_DRV_Init\(\)](#) function. This allows the user to configure settings of the EDMA peripheral with a single function call.

Data Fields

- [edma_channel_arbitration_t chnArbitration](#)
eDMA channel arbitration.

10.3.8.1.0.24 Field Documentation

10.3.8.1.0.24.1 [edma_channel_arbitration_t edma_user_config_t::chnArbitration](#)

10.3.8.2 struct [edma_chn_state_t](#)

Data Fields

- uint8_t [channel](#)
Virtual channel indicator.
- [edma_callback_t callback](#)
Callback function pointer for the eDMA channel.
- void * [parameter](#)
Parameter for the callback function pointer.
- volatile [edma_chn_status_t status](#)
eDMA channel status.

10.3.8.2.0.25 Field Documentation

10.3.8.2.0.25.1 [uint8_t edma_chn_state_t::channel](#)

10.3.8.2.0.25.2 [edma_callback_t edma_chn_state_t::callback](#)

It will be called at the eDMA channel complete and eDMA channel error.

eDMA Peripheral driver

10.3.8.2.0.25.3 void* edma_chn_state_t::parameter

10.3.8.2.0.25.4 volatile edma_chn_status_t edma_chn_state_t::status

10.3.8.3 struct edma_scatter_gather_list_t

Data Fields

- uint32_t **address**
Address of buffer.
- uint32_t **length**
Length of buffer.

10.3.8.3.0.26 Field Documentation

10.3.8.3.0.26.1 uint32_t edma_scatter_gather_list_t::address

10.3.8.3.0.26.2 uint32_t edma_scatter_gather_list_t::length

10.3.8.4 struct edma_state_t

This structure holds data that is used by the eDMA peripheral driver to manage multi eDMA channels. The user passes the memory for this run-time state structure and the eDMA driver populates the members.

Data Fields

- **edma_chn_state_t *volatile chn [FSL_FEATURE_EDMA_DMAMUX_CHANNELS]**
Pointer array storing channel state.

10.3.8.4.0.27 Field Documentation

10.3.8.4.0.27.1 edma_chn_state_t* volatile edma_state_t::chn[FSL_FEATURE_EDMA_DMAMUX_CHANNELS]

10.3.9 Macro Definition Documentation

10.3.9.1 #define STCD_SIZE(*number*) ((*number* + 1) * 32)

Software TCD is aligned to 32 bytes. To make sure the software TCD can meet the eDMA module requirement, allocate memory with extra 32 bytes.

- 10.3.9.2 `#define VIRTUAL_CHN_TO_EDMA_MODULE_REGBASE(chn) g_edmaBaseAddr[chn/FSL_FEATURE_EDMA_MODULE_CHANNEL]`
- 10.3.9.3 `#define VIRTUAL_CHN_TO_EDMA_CHN(chn) (chn%FSL_FEATURE_EDMA_MODULE_CHANNEL)`
- 10.3.9.4 `#define VIRTUAL_CHN_TO_DMAMUX_MODULE_REGBASE(chn) g_dmamuxBaseAddr[chn/FSL_FEATURE_DMAMUX_MODULE_CHANNEL]`
- 10.3.9.5 `#define VIRTUAL_CHN_TO_DMAMUX_CHN(chn) (chn%FSL_FEATURE_DMAMUX_MODULE_CHANNEL)`

10.3.10 Typedef Documentation

10.3.10.1 `typedef void(* edma_callback_t)(void *parameter, edma_chn_status_t status)`

Prototype for the callback function registered in the eDMA driver.

10.3.11 Enumeration Type Documentation

10.3.11.1 `enum edma_chn_status_t`

A structure describing the eDMA channel status. The user can get the status by callback parameter or by calling EDMA_DRV_getStatus() function.

Enumerator

kEDMAChnNormal eDMA channel is occupied.

kEDMAChnIdle eDMA channel is idle.

kEDMAChnError An error occurs in the eDMA channel.

10.3.11.2 `enum edma_chn_state_type_t`

Enumerator

kEDMAInvalidChannel Macros indicate the failure of the channel request.

kEDMAAnyChannel Macros used when requesting channel dynamically.

10.3.11.3 `enum edma_transfer_type_t`

Enumerator

kEDMAPeripheralToMemory Transfer from peripheral to memory.

eDMA Peripheral driver

kEDMAMemoryToPeripheral Transfer from memory to peripheral.

kEDMAMemoryToMemory Transfer from memory to memory.

10.3.12 Function Documentation

10.3.12.1 `edma_status_t EDMA_DRV_Init (edma_state_t * edmaState, const edma_user_config_t * userConfig)`

This function initializes the run-time state structure to provide the eDMA channel allocation release, protect, and track the state for channels. This function also opens the clock to the eDMA modules, resets the eDMA modules, initializes the module to user-defined settings and default settings. This is an example to set up the `edma_state_t` and the `edma_user_config_t` parameters and to call the `EDMA_DRV_Init` function by passing in these parameters.

```
edma_state_t state;      <- The user simply allocates memory for this structure.  
edma_user_config_t userConfig;    <- The user fills out members for this structure.  
  
userConfig.chnArbitration = kEDMACHnArbitrationRoundrobin;  
#if (FSL_FEATURE_EDMA_CHANNEL_GROUP_COUNT > 0x1U)  
    //configuration for 2 lines below only valid for SoCs with more than one group.  
    userConfig.groupArbitration = kEDMAGroupArbitrationFixedPriority;  
    userConfig.groupPriority = kEDMAGroup0PriorityHighGroup1PriorityLow;  
#endif  
userConfig.notHaltOnError = false;    <- The default setting is false, means eDMA halt on error.  
  
EDMA_DRV_Init (&state, &userConfig);
```

Parameters

<code>edmaState</code>	The pointer to the eDMA peripheral driver state structure. The user passes the memory for this run-time state structure and the eDMA peripheral driver populates the members. This run-time state structure keeps track of the eDMA channels status. The memory must be kept valid before calling the <code>EDMA_DRV_DeInit</code> .
<code>userConfig</code>	User configuration structure for eDMA peripheral drivers. The user populates the members of this structure and passes the pointer of this structure into the function.

Returns

An eDMA error codes or `kStatus_EDMA_Success`.

10.3.12.2 `edma_status_t EDMA_DRV_Deinit (void)`

This function resets the eDMA modules to reset state, gates the clock, and disables the interrupt to the core.

Returns

An eDMA error codes or `kStatus_EDMA_Success`.

10.3.12.3 `uint8_t EDMA_DRV_RequestChannel (uint8_t channel, dma_request_source_t source, edma_chn_state_t * chn)`

This function allocates eDMA channel according to the required channel allocation and corresponding to the eDMA hardware request, initializes the channel state memory provided by user and fills out the members. This functions also sets up the hardware request configuration according to the user's requirements.

For Kinetis devices, a hardware request can be mapped to eDMA channels and used for the channel trigger. Some hardware requests can only be mapped to a limited channels. For example, the Kinetis K70FN1-M0VMJ15 device eDMA module has 2 eDMA channel groups. The first group consists of the channel 0 - 15. The second group consists of channel 16 - 31. The hardware request UART0-Receive can be only mapped to group 1. Therefore, the hardware request is one of the parameter that the user needs to provide for the channel request. Channel needn't be triggered by the peripheral hardware request. The user can provide the ALWAYSON type hardware request to trigger the channel continuously.

This function provides two ways to allocate an eDMA channel: statically and dynamically. In a static allocation, the user provides the required channel number and eDMA driver tries to allocate the required channel to the user. If the channel is not occupied, the eDMA driver is successfully assigned to the user. If the channel is already occupied, the user gets the return value kEDMAInvalidChn. This is an example to request a channel statically:

```
uint32_t channelNumber = 14;    <- Try to allocate the channel 14
edma_chn_state_t chn;        <- The user simply allocates memory for this structure.

if ( kEDMAInvalidChannel == EDMA_DRV_RequestChannel(channel,
            kDmaRequestMux0AlwaysOn54, chn))
{
    printf("request channel %d failed!\n", channel);
}
```

In a dynamic allocation, any of the free eDMA channels are available for use. eDMA driver assigns the first free channel to the user. This is an example for user to request a channel dynamically :

```
uint32_t channel;      <- Store the allocated channel number.
edma_chn_state_t chn;    <- The user simply allocates memory for this structure.

channel = EDMA_DRV_RequestChannel(kEDMAAnyChannel,
            kDmaRequestMux0AlwaysOn54, chn);

if (channel == kEDMAInvalidChannel)
{
    printf("request channel %d failed!\n", channel);
}
else
{
    printf("Channel %d is successfully allocated! /n", channel);
}
```

eDMA Peripheral driver

Parameters

<i>channel</i>	Requested channel number. If the channel is assigned with the kEDMAAnyChannel, the eDMA driver allocates the channel dynamically. If the channel is assigned with a valid channel number, the eDMA driver allocates that channel.
<i>source</i>	eDMA hardware request number.
<i>chn</i>	The pointer to the eDMA channel state structure. The user passes the memory for this run-time state structure. The eDMA peripheral driver populates the members. This run-time state structure keeps tracks of the eDMA channel status. The memory must be kept valid before calling the EDMA_DRV_ReleaseChannel() .

Returns

Successfully allocated channel number or the kEDMAInvalidChannel indicating that the request is failed.

10.3.12.4 **edma_status_t EDMA_DRV_ReleaseChannel (edma_chn_state_t * *chn*)**

This function stops the eDMA channel and disables the interrupt of this channel. The channel state structure can be released after this function is called.

Parameters

<i>chn</i>	The pointer to the channel state structure.
------------	---

Returns

An eDMA error codes or kStatus_EDMA_Success.

10.3.12.5 **static edma_status_t EDMA_DRV_PrepDescriptorTransfer (edma_chn_state_t * *chn*, edma_software_tcd_t * *stcd*, edma_transfer_config_t * *config*, bool *enableInt*, bool *disableDmaRequest*) [inline], [static]**

This function sets up the basic transfer for the descriptor. The minor loop setting is not used because the minor loop configuration impacts the global eDMA setting. The source minor loop offset is relevant to the destination minor loop offset. For these reasons, the minor loop offset configuration is treated as an advanced configuration. The user can call the [EDMA_HAL_STCDSetMinorLoopOffset\(\)](#) function to configure the minor loop offset feature.

Parameters

<i>channel</i>	Virtual channel number.
<i>chn</i>	The pointer to the channel state structure.
<i>stcd</i>	The pointer to the descriptor.
<i>config</i>	Configuration for the basic transfer.
<i>enableInt</i>	Enables (true) or Disables (false) interrupt on TCD complete.
<i>disableDmaRequest</i>	Disables (true) or Enable (false) DMA request on TCD complete.

10.3.12.6 static edma_status_t EDMA_DRV_PrepDescriptorScatterGather (edma_software_tcd_t * *stcd*, edma_software_tcd_t * *nextStcd*) [inline], [static]

This function enables the scatter/gather feature for the software TCD and configures the next TCD address. This address points to the beginning of a 0-modulo-32 byte region containing the next transfer TCD to be loaded into this channel. The channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte. Otherwise, a configuration error is reported.

Parameters

<i>stcd</i>	The pointer to the software TCD, which needs to link to the software TCD. The address needs to be aligned to 32 bytes.
<i>nextStcd</i>	The pointer to the software TCD, which is to be linked to the software TCD. The address needs to be aligned to 32 bytes.

10.3.12.7 static edma_status_t EDMA_DRV_PrepDescriptorChannelLink (edma_software_tcd_t * *stcd*, uint32_t *linkChn*) [inline], [static]

If the major link is enabled, after the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel start bits.

Parameters

<i>stcd</i>	The pointer to the software TCD. The address need to be aligned to 32 bytes.
-------------	--

eDMA Peripheral driver

<i>linkChn</i>	Channel number for major link
----------------	-------------------------------

10.3.12.8 **edma_status_t EDMA_DRV_PushDescriptorToReg (edma_chn_state_t * *chn*, edma_software_tcd_t * *stcd*)**

Parameters

<i>chn</i>	The pointer to the channel state structure.
<i>stcd</i>	memory pointing to the software TCD.

10.3.12.9 **edma_status_t EDMA_DRV_ConfigLoopTransfer (edma_chn_state_t * *chn*, edma_software_tcd_t * *stcd*, edma_transfer_type_t *type*, uint32_t *srcAddr*, uint32_t *destAddr*, uint32_t *size*, uint32_t *bytesOnEachRequest*, uint32_t *totalLength*, uint8_t *number*)**

This function configures the descriptors in a loop chain. The user passes a block of memory into this function and the memory is divided into the "period" sub blocks. The DMA driver configures the "period" descriptors. Each descriptor stands for a sub block. The DMA driver transfers data from the first descriptor to the last descriptor. Then, the DMA driver wraps to the first descriptor to continue the loop. The interrupt handler is called every time a descriptor is completed. The user can get a transfer status of a descriptor by calling the `edma_get_descriptor_status()` function in the interrupt handler or any other task context. At the same time, calling the `edma_update_descriptor()` function notifies the DMA driver that the content belonging to a descriptor is already updated and the DMA needs to count it as an underflow next time it loops to this descriptor.

Parameters

<i>chn</i>	The pointer to the channel state structure.
<i>stcd</i>	Memory pointing to software TCDs. The user must prepare this memory block. The required memory size is equal to a "period" * size of(edma_software_tcd_t). At the same time, the "stcd" must align with 32 bytes. If not, an error occurs in the eDMA driver.

<i>type</i>	Transfer type.
<i>srcAddr</i>	A source register address or a start memory address.
<i>destAddr</i>	A destination register address or a start memory address.
<i>size</i>	Size to be transferred on every DMA write/read. Source/Dest share the same write/read size.
<i>bytesOnEachRequest</i>	Size write/read for every trigger of the DMA request.
<i>totalLength</i>	Total length of Memory.
<i>number</i>	A number of the descriptor that is configured for this transfer configuration.

Returns

An error code of kStatus_EDMA_Success

10.3.12.10 `edma_status_t EDMA_DRV_ConfigScatterGatherTransfer (edma_chn_state_t * chn, edma_software_tcd_t * stcd, edma_transfer_type_t type, uint32_t size, uint32_t bytesOnEachRequest, edma_scatter_gather_list_t * srcList, edma_scatter_gather_list_t * destList, uint8_t number)`

This function configures the descriptors into a sing-end chain. The user passes blocks of memory into this function. The interrupt is triggered only when the last memory block is completed. The memory block information is passed with the `edma_scatter_gather_list_t` data structure, which can tell the memory address and length. The DMA driver configures the descriptor for each memory block, transfers the descriptor from the first one to the last one, and stops.

Parameters

<i>chn</i>	The pointer to the channel state structure.
<i>stcd</i>	Memory pointing to software TCDs. The user must prepare this memory block. The required memory size is equal to the "number" * size of(<code>edma_software_tcd_t</code>). At the same time, the "stcd" must align with 32 bytes. If not, an error occurs in the eDMA driver.

eDMA Peripheral driver

<i>type</i>	Transfer type.
<i>size</i>	Size to be transferred on each DMA write/read. Source/Dest share the same write/read size.
<i>bytesOnEach-Request</i>	Size write/read for each trigger of the DMA request.
<i>srcList</i>	Data structure storing the address and length to be transferred for source memory blocks. If the source memory is peripheral, the length is not used.
<i>destList</i>	Data structure storing the address and length to be transferred for destination memory blocks. If in the memory-to-memory transfer mode, the user must ensure that the length of the destination scatter gather list is equal to the source scatter gather list. If the destination memory is a peripheral register, the length is not used.
<i>number</i>	A number of memory block contained in the scatter gather list.

Returns

An error code of kStatus_EDMA_Success

10.3.12.11 edma_status_t EDMA_DRV_StartChannel (edma_chn_state_t * *chn*)

This function enables the eDMA channel DMA request.

Parameters

<i>chn</i>	The pointer to the channel state structure.
------------	---

Returns

An eDMA error codes or kStatus_EDMA_Success.

10.3.12.12 edma_status_t EDMA_DRV_StopChannel (edma_chn_state_t * *chn*)

This function disables the eDMA channel DMA request.

Parameters

<i>chn</i>	The pointer to the channel state structure.
------------	---

Returns

An eDMA error codes or kStatus_EDMA_Success.

10.3.12.13 `edma_status_t EDMA_DRV_InstallCallback (edma_chn_state_t * chn, edma_callback_t callback, void * parameter)`

This function registers the callback function and the parameter into the eDMA channel state structure. The callback function is called when the channel is complete or a channel error occurs. The eDMA driver passes the channel status to this callback function to indicate whether it is caused by the channel complete event or the channel error event.

To un-register the callback function, set the callback function to "NULL" and call this function.

Parameters

<i>chn</i>	The pointer to the channel state structure.
<i>callback</i>	The pointer to the callback function.
<i>parameter</i>	The pointer to the callback function's parameter.

Returns

An eDMA error codes or kStatus_EDMA_Success.

10.3.12.14 `void EDMA_DRV_IRQHandler (uint8_t channel)`

This function is provided as the default flow for eDMA channel interrupt. This function clears the status and calls the callback functions. The user can add this function into the hardware interrupt entry and implement a custom interrupt action function.

Parameters

<i>channel</i>	Virtual channel number.
----------------	-------------------------

10.3.12.15 `void EDMA_DRV_ErrorIRQHandler (uint8_t instance)`

This function is provided as the default action for eDMA module error interrupt. This function clears status, stops the error on a eDMA channel, and calls the eDMA channel callback function if the error eDMA channel is already requested. The user can add this function into the eDMA error interrupt entry and implement a custom interrupt action function.

Parameters

eDMA Peripheral driver

<i>instance</i>	eDMA module indicator.
-----------------	------------------------

10.3.12.16 static edma_chn_status_t EDMA_DRV_GetChannelStatus (edma_chn_state_t * *chn*) [inline], [static]

Parameters

<i>chn</i>	The pointer to the channel state structure.
------------	---

Returns

Channel status.

10.3.12.17 static uint32_t EDMA_DRV_GetFinishedBytes (edma_chn_state_t * *chn*) [inline], [static]

This function checks the TCD (Task Control Descriptor) status for a specified eDMA channel and returns the bytes that remain to the user. This function can only be used for one TCD scenario.

Parameters

<i>chn</i>	The pointer to the channel state structure.
------------	---

Returns

Bytes already transferred for the current TCD.

10.3.13 Variable Documentation

10.3.13.1 const uint32_t g_edmaBaseAddr[]

10.3.13.2 const uint32_t g_dmamuxBaseAddr[]

10.3.13.3 const IRQn_Type g_edmaIrqlId[HW_DMA_INSTANCE_COUNT][FSL_FEATURE_EDMA_MODULE_CHANNEL]

10.3.13.4 const IRQn_Type g_edmaErrIrqlId[HW_DMA_INSTANCE_COUNT]

10.4 eDMA request

10.4.1 Overview

This section describes the programming interface of the eDMA DMA request resource.

Enumerations

- enum `dma_request_source_t`
Structure for the DMA hardware request.
- enum `dma_request_source_t`
Structure for the DMA hardware request.

10.4.2 Enumeration Type Documentation

10.4.2.1 enum `dma_request_source_t`

Defines the structure for the DMA hardware request collections. The user can configure the hardware request into DMAMUX to trigger the DMA transfer accordingly. The index of the hardware request varies according to the to SoC.

10.4.2.2 enum `dma_request_source_t`

Defines the structure for the DMA hardware request collections. The user can configure the hardware request into DMAMUX to trigger the DMA transfer accordingly. The index of the hardware request varies according to the to SoC.

Chapter 11

Ethernet MAC (ENET)

11.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Ethernet (ENET) block of Kinetis devices.

Modules

- ENET HAL driver

11.2 ENET HAL driver

11.2.1 Overview

This section describes the programming interface of the ENET HAL driver.

Data Structures

- struct [enet_bd_struct_t](#)
Defines the buffer descriptor structure for the little-Endian system and endianness configurable IP. [More...](#)
- struct [enet_config_rmii_t](#)
Defines the RMII/MII configuration structure. [More...](#)
- struct [enet_config_ptp_timer_t](#)
Defines the configuration structure for the 1588 PTP timer. [More...](#)
- struct [enet_config_tx_fifo_t](#)
Defines the transmit FIFO configuration. [More...](#)
- struct [enet_config_rx_fifo_t](#)
Defines the receive FIFO configuration. [More...](#)
- struct [enet_mib_rx_stat_t](#)
@ brief Defines the receive statistics of MIB [More...](#)
- struct [enet_mib_tx_stat_t](#)
@ brief Defines the transmit statistics of MIB [More...](#)
- struct [enet_special_maccfg_t](#)
Define the special configure for Rx and Tx controller. [More...](#)
- struct [enet_mac_config_t](#)
Defines the basic configuration structure for the ENET device. [More...](#)

Macros

- #define [SYSTEM_LITTLE_ENDIAN](#) (1)
Defines the system endian type.
- #define [BSWAP_16](#)(x) ((uint16_t)((uint16_t)((uint16_t)(x) & (uint16_t)0xFF00) >> 0x8) | (uint16_t)((((uint16_t)(x) & (uint16_t)0xFF) << 0x8)))
Define macro to do the endianness swap.
- #define [ENET_ALIGN](#)(x, align) (((unsigned int)((x) + ((align)-1)) & (unsigned int)(~(unsigned int)((align)- 1))))
Defines the alignment operation.
- #define [BD_SHORTSWAP](#)(n) (n)
Defines the macro used for byte order change on Buffer descriptor.

Enumerations

- enum `enet_status_t` { ,

kStatus_ENET_InvalidInput,

kStatus_ENET_InvalidDevice,

kStatus_ENET_InitTimeout,

kStatus_ENET_MemoryAllocateFail,

kStatus_ENET_GetClockFreqFail,

kStatus_ENET_Initialized,

kStatus_ENET_Open,

kStatus_ENET_Close,

kStatus_ENET_Layer2UnInitialized,

kStatus_ENET_Layer2OverLarge,

kStatus_ENET_Layer2BufferFull,

kStatus_ENET_Layer2TypeError,

kStatus_ENET_PtpringBufferFull,

kStatus_ENET_PtpringBufferEmpty,

kStatus_ENET_SMIUninitialized,

kStatus_ENET_SMIVisitTimeout,

kStatus_ENET_RxbdInvalid,

kStatus_ENET_RxbdEmpty,

kStatus_ENET_RxbdTrunc,

kStatus_ENET_RxbdError,

kStatus_ENET_RxBdFull,

kStatus_ENET_SmallRxBuffSize,

kStatus_ENET_NoEnoughRxBuffers,

kStatus_ENET_LargeBufferFull,

kStatus_ENET_TxLarge,

kStatus_ENET_TxbdFull,

kStatus_ENET_TxbdNull,

kStatus_ENET_TxBufferNull,

kStatus_ENET_NoRxBufferLeft,

kStatus_ENET_UnknownCommand,

kStatus_ENET_TimeOut,

kStatus_ENET_MulticastPointerNull,

kStatus_ENET_NoMulticastAddr,

kStatus_ENET_AlreadyAddedMulticast,

kStatus_ENET_PHYAutoDiscoverFail }
- Defines the Status return codes.*
- enum `enet_rx_bd_control_status_t` {

ENET HAL driver

```
kEnetRxBdEmpty = 0x8000U,  
kEnetRxBdRxSoftOwner1 = 0x4000U,  
kEnetRxBdWrap = 0x2000U,  
kEnetRxBdRxSoftOwner2 = 0x1000U,  
kEnetRxBdLast = 0x0800U,  
kEnetRxBdMiss = 0x0100U,  
kEnetRxBdBroadCast = 0x0080U,  
kEnetRxBdMultiCast = 0x0040U,  
kEnetRxBdLengthViolation = 0x0020U,  
kEnetRxBdNoOctet = 0x0010U,  
kEnetRxBdCrc = 0x0004U,  
kEnetRxBdOverRun = 0x0002U,  
kEnetRxBdTrunc = 0x0001U }
```

Defines the control and status region of the receive buffer descriptor.

- enum `enet_rx_bd_control_extend0_t` {
 kEnetRxBdIpv4 = 0x0001U,
 kEnetRxBdIpv6 = 0x0002U,
 kEnetRxBdVlan = 0x0004U,
 kEnetRxBdProtocolChecksumErr = 0x0010U,
 kEnetRxBdIpHeaderChecksumErr = 0x0020U }

Defines the control extended region1 of the receive buffer descriptor.

- enum `enet_rx_bd_control_extend1_t` {
 kEnetRxBdIntrrupt = 0x0080U,
 kEnetRxBdUnicast = 0x0100U,
 kEnetRxBdCollision = 0x0200U,
 kEnetRxBdPhyErr = 0x0400U,
 kEnetRxBdMacErr = 0x8000U }

Defines the control extended region2 of the receive buffer descriptor.

- enum `enet_tx_bd_control_status_t` {
 kEnetTxBdReady = 0x8000U,
 kEnetTxBdTxSoftOwner1 = 0x4000U,
 kEnetTxBdWrap = 0x2000U,
 kEnetTxBdTxSoftOwner2 = 0x1000U,
 kEnetTxBdLast = 0x0800U,
 kEnetTxBdTransmitCrc = 0x0400U }

Defines the control status of the transmit buffer descriptor.

- enum `enet_tx_bd_control_extend0_t` {
 kEnetTxBdTxErr = 0x8000U,
 kEnetTxBdTxUnderFlowErr = 0x2000U,
 kEnetTxBdExcessCollisionErr = 0x1000U,
 kEnetTxBdTxFrameErr = 0x0800U,
 kEnetTxBdLatecollisionErr = 0x0400U,
 kEnetTxBdOverFlowErr = 0x0200U,
 kEnetTxTimestampErr = 0x0100U }

Defines the control extended region1 of the transmit buffer descriptor.

- enum `enet_tx_bd_control_extend1_t` {

```
kEnetTxBdTxInterrupt = 0x4000U,
kEnetTxBdTimeStamp = 0x2000U }
```

Defines the control extended region2 of the transmit buffer descriptor.

- enum `enet_constant_parameter_t` {

kEnetMacAddrLen = 6U,
 kEnetHashValMask = 0x1FU,
 kEnetMinBuffSize = 256U,
 kEnetMaxTimeout = 0xFFFFU,
 kEnetMdcFreq = 2500000U }

Defines the macro to the different ENET constant value.

- enum `enet_fifo_configure_t` {

kEnetMinTxFifoAlmostFull = 6U,
 kEnetMinFifoAlmostEmpty = 4U,
 kEnetDefaultTxFifoAlmostFull = 8U }

Defines the normal fifo configuration for ENET MAC.

- enum `enet_mac_operate_mode_t` {

kEnetMacNormalMode = 0U,
 kEnetMacSleepMode = 1U }

Defines the normal operating mode and sleep mode for ENET MAC.

- enum `enet_config_rmii_mode_t` {

kEnetCfgMii = 0U,
 kEnetCfgRmii = 1U }

Defines the RMII or MII mode for data interface between the MAC and the PHY.

- enum `enet_config_speed_t` {

kEnetCfgSpeed100M = 0U,
 kEnetCfgSpeed10M = 1U }

Defines the 10 Mbps or 100 Mbps speed mode for the data transfer.

- enum `enet_config_duplex_t` {

kEnetCfgHalfDuplex = 0U,
 kEnetCfgFullDuplex = 1U }

Defines the half or full duplex mode for the data transfer.

- enum `enet_mii_write_t` {

kEnetWriteNoCompliant = 0U,
 kEnetWriteValidFrame = 1U }

Defines the write operation for the MII.

- enum `enet_mii_read_t` {

kEnetReadValidFrame = 2U,
 kEnetReadNoCompliant = 3U }

Defines the read operation for the MII.

- enum `enet_special_address_filter_t` {

kEnetSpecialAddressInit = 0U,
 kEnetSpecialAddressEnable = 1U,
 kEnetSpecialAddressDisable = 2U }

Defines the initialization, enables or disables the operation for a special address filter.

- enum `enet_timer_channel_t` {

ENET HAL driver

```
kEnetTimerChannel1 = 0U,  
kEnetTimerChannel2 = 1U,  
kEnetTimerChannel3 = 2U,  
kEnetTimerChannel4 = 3U }
```

Defines the 1588 timer channel numbers.

- enum `enet_timer_channel_mode_t` {
 kEnetChannelDisable = 0U,
 kEnetChannelRisingCapture = 1U,
 kEnetChannelFallingCapture = 2U,
 kEnetChannelBothCapture = 3U,
 kEnetChannelSoftCompare = 4U,
 kEnetChannelToggleCompare = 5U,
 kEnetChannelClearCompare = 6U,
 kEnetChannelSetCompare = 7U,
 kEnetChannelClearCompareSetOverflow = 10U,
 kEnetChannelSetCompareClearOverflow = 11U,
 kEnetChannelPulseLowonCompare = 14U,
 kEnetChannelPulseHighonCompare = 15U }

Defines the capture or compare mode for 1588 timer channels.

- enum `enet_interrupt_request_t` {
 kEnetBabrInterrupt = 0x40000000U,
 kEnetBabtInterrupt = 0x20000000U,
 kEnetGraceStopInterrupt = 0x10000000U,
 kEnetTxFrameInterrupt = 0x08000000U,
 kEnetTxByteInterrupt = 0x04000000U,
 kEnetRxFrameInterrupt = 0x02000000U,
 kEnetRxByteInterrupt = 0x01000000U,
 kEnetMiiInterrupt = 0x00800000U,
 kEnetEBusERInterrupt = 0x00400000U,
 kEnetLateCollisionInterrupt = 0x00200000U,
 kEnetRetryLimitInterrupt = 0x00100000U,
 kEnetUnderrunInterrupt = 0x00080000U,
 kEnetPayloadRxInterrupt = 0x00040000U,
 kEnetWakeupInterrupt = 0x00020000U,
 kEnetTsAvailInterrupt = 0x00010000U,
 kEnetTsTimerInterrupt = 0x00008000U,
 kEnetAllInterrupt = 0x7FFFFFFFU }

Defines the RXFRAME/RXBYTE/TXFRAME/TXBYTE/MII/TSTIMER/TSAVAIL interrupt source for ENET.
T.

- enum `enet_irq_number_t` {
 kEnetTsTimerNumber = 0,
 kEnetReceiveNumber = 1,
 kEnetTransmitNumber = 2,
 kEnetMiiErrorNumber = 3 }
- enum `enet_frame_max_t` {

```

kEnetNsecOneSec = 1000000000,
kEnetMaxFrameSize = 1518,
kEnetMaxFrameVlanSize = 1522,
kEnetMaxFrameDateSize = 1500,
kEnetDefaultTruncLen = 2047,
kEnetDefaultIpg = 12,
kEnetMaxValidTxIpg = 27,
kEnetMinValidTxIpg = 8,
kEnetMaxMdioHoldCycle = 7,
kEnetMaxFrameBdNumbers = 6,
kEnetFrameFcsLen = 4,
kEnetEthernetHeadLen = 14,
kEnetEthernetVlanHeadLen = 18 }

```

Defines the ENET main constant.

- enum `enet_txaccelerator_config_t` {

kEnetTxAccelIsShift16Enabled = 0x01U,

kEnetTxAccelIpCheckEnabled = 0x08U,

kEnetTxAccelProtoCheckEnabled = 0x10U }

Defines the transmit accelerator configuration.

- enum `enet_rxaccelerator_config_t` {

kEnetRxAccelPadRemoveEnabled = 0x01U,

kEnetRxAccelIpCheckEnabled = 0x02U,

kEnetRxAccelProtoCheckEnabled = 0x04U,

kEnetRxAccelMacCheckEnabled = 0x40U,

kEnetRxAccelIsShift16Enabled = 0x80U }

Defines the receive accelerator configuration.

- enum `enet_mac_control_flag_t` {

kEnetStopModeEnable = 0x1U ,

kEnetPayloadlenCheckEnable = 0x4U,

kEnetRxFlowControlEnable = 0x8U,

kEnetRxCrcFwdEnable = 0x10U,

kEnetRxPauseFwdEnable = 0x20U,

kEnetRxPadRemoveEnable = 0x40U,

kEnetRxBcRejectEnable = 0x80U,

kEnetRxPromiscuousEnable = 0x100U,

kEnetTxCrcFwdEnable = 0x200U,

kEnetTxCrcBdEnable = 0x400U,

kEnetMacAddrInsert = 0x800U,

kEnetTxAccelEnable = 0x1000U,

kEnetRxAccelEnable = 0x2000U,

kEnetStoreAndFwdEnable = 0x4000U,

kEnetMacMibEnable = 0x8000U,

kEnetSMIPreambleDisable = 0x10000U,

kEnetVlanTagEnabled = 0x20000U,

kEnetMacEnhancedEnable = 0x40000U }

ENET HAL driver

Defines the ENET MAC control Configure.

Functions

- `uint32_t ENET_HAL_Init (uint32_t baseAddr)`
Initializes the ENET module to reset status.
- `void ENET_HAL_SetMac (uint32_t baseAddr, const enet_mac_config_t *macCfgPtr, uint32_t sysClk)`
Configures the Mac controller of the ENET device.
- `void ENET_HAL_SetTxBuffDescriptors (uint32_t baseAddr, volatile enet_bd_struct_t *txBds, uint8_t *txBuffer, uint32_t txBdNumber, uint32_t txBuffSizeAlign)`
Configures the ENET transmit buffer descriptors.
- `void ENET_HAL_SetRxBuffDescriptors (uint32_t baseAddr, volatile enet_bd_struct_t *rxBds, uint8_t *rxBuffer, uint32_t rxBdNumber, uint32_t rxBuffSizeAlign)`
Configures the ENET receive buffer descriptors.
- `void ENET_HAL_SetFifo (uint32_t baseAddr, const enet_mac_config_t *macCfgPtr)`
Configures the transmit and receive FIFO of the ENET device.
- `void ENET_HAL_GetMibRxStat (uint32_t baseAddr, enet_mib_rx_stat_t *rxStat)`
Gets all received statistics from MIB.
- `void ENET_HAL_GetMibTxStat (uint32_t baseAddr, enet_mib_tx_stat_t *txStat)`
Gets all transmitted statistics from MIB.
- `static void ENET_HAL_SetStopCmd (uint32_t baseAddr, bool enable)`
Enables or disables the stop enable signal control.
- `static void ENET_HAL_SetDebugCmd (uint32_t baseAddr, bool enable)`
Enables or disables Mac entering the hardware freeze mode when the device enters debug mode.
- `void ENET_HAL_SetMacMode (uint32_t baseAddr, enet_mac_operate_mode_t mode)`
Configures the Mac operating mode.
- `void ENET_HAL_SetMacAddr (uint32_t baseAddr, uint8_t *hwAddr)`
Sets the Mac address.
- `static void ENET_HAL_SetMacAddrInsertCmd (uint32_t baseAddr, bool enable)`
Enables or disables Mac address modification on transmit.
- `void ENET_HAL_SetMulticastAddrHash (uint32_t baseAddr, uint32_t crcValue, enet_special_address_filter_t mode)`
Sets the hardware addressing filtering to a multicast group address.
- `void ENET_HAL_SetUnicastAddrHash (uint32_t baseAddr, uint32_t crcValue, enet_special_address_filter_t mode)`
Sets the hardware addressing filtering to an individual address.
- `static void ENET_HAL_SetTxcrcFwdCmd (uint32_t baseAddr, bool enable)`
Enables/disables the forwarding frame from an application with the CRC for the transmitted frames.
- `static void ENET_HAL_SetRxcrcFwdCmd (uint32_t baseAddr, bool enable)`
Enables/disables forward the CRC field of the received frame.
- `static void ENET_HAL_SetPauseFwdCmd (uint32_t baseAddr, bool enable)`
Enables/disables pause frames forwarding.
- `static void ENET_HAL_SetPadRemoveCmd (uint32_t baseAddr, bool enable)`
Enables/disables frame padding remove on receive.
- `static void ENET_HAL_SetFlowControlCmd (uint32_t baseAddr, bool enable)`
Enables/disables the flow control.
- `static void ENET_HAL_SetBroadcastRejectCmd (uint32_t baseAddr, bool enable)`
Enables/disables the broadcast frame reject.

- static void [ENET_HAL_SetPayloadCheckCmd](#) (uint32_t baseAddr, bool enable)
Enables/disables the payload length check.
- static void [ENET_HAL_SetGraceTxStopCmd](#) (uint32_t baseAddr, bool enable)
Enables/disables the graceful transmit stop.
- static void [ENET_HAL_SetPauseDuration](#) (uint32_t baseAddr, uint32_t pauseDuration)
Sets the pause duration for the pause frame.
- static void [ENET_HAL_SetTxPauseCmd](#) (uint32_t baseAddr, bool enable)
Configures the pause duration field and transmits the pause frame.
- static bool [ENET_HAL_GetTxPause](#) (uint32_t baseAddr)
Gets the transmit pause frame status.
- static bool [ENET_HAL_GetRxPause](#) (uint32_t baseAddr)
Gets the receive pause frame status.
- void [ENET_HAL_SetTxInterPacketGap](#) (uint32_t baseAddr, uint32_t ipgValue)
Sets the transmit inter-packet gap.
- static void [ENET_HAL_SetTruncLen](#) (uint32_t baseAddr, uint32_t length)
Sets the receive frame truncation length.
- static void [ENET_HAL_SetRxMaxFrameLen](#) (uint32_t baseAddr, uint32_t maxFrameSize)
Sets the maximum receive frame length.
- static uint16_t [ENET_HAL_GetRxMaxFrameLen](#) (uint32_t baseAddr)
Gets the maximum receive frame length.
- static void [ENET_HAL_SetRxMaxBuffSize](#) (uint32_t baseAddr, uint32_t maxBufferSize)
Sets the maximum receive buffer size for receive buffer descriptor.
- void [ENET_HAL_SetTxFifo](#) (uint32_t baseAddr, [enet_config_tx_fifo_t](#) *thresholdCfg)
Configures the ENET transmit FIFO.
- void [ENET_HAL_SetRxFifo](#) (uint32_t baseAddr, [enet_config_rx_fifo_t](#) *thresholdCfg)
Configures the ENET receive FIFO.
- static void [ENET_HAL_SetRxBuffDescripAddr](#) (uint32_t baseAddr, uint32_t rxBdAddr)
Sets the start address for ENET receive buffer descriptors.
- static void [ENET_HAL_SetTxBuffDescripAddr](#) (uint32_t baseAddr, uint32_t txBdAddr)
Sets the start address for ENET transmit buffer descriptors.
- void [ENET_HAL_InitRxBuffDescriptors](#) (volatile [enet_bd_struct_t](#) *rxBds, uint8_t *rxBuff, uint32_t rxbdNum, uint32_t rxBuffSizeAlign)
Initializes receive buffer descriptors.
- void [ENET_HAL_InitTxBuffDescriptors](#) (volatile [enet_bd_struct_t](#) *txBds, uint8_t *txBuff, uint32_t txbdNum, uint32_t txBuffSizeAlign)
Initializes transmit buffer descriptors.
- void [ENET_HAL_UpdateRxBuffDescriptor](#) (volatile [enet_bd_struct_t](#) *rxBds, uint8_t *data, bool isbufferUpdate)
Updates the receive buffer descriptor.
- void [ENET_HAL_UpdateTxBuffDescriptor](#) (volatile [enet_bd_struct_t](#) *txBds, uint16_t length, bool isTxtsCfged, bool isTxCrcEnable, bool isLastOne)
Updates the transmit buffer descriptor.
- static void [ENET_HAL_ClearTxBuffDescriptor](#) (volatile [enet_bd_struct_t](#) *curBd)
Clears the context in the transmit buffer descriptors.
- static uint16_t [ENET_HAL_GetRxBuffDescripControl](#) (volatile [enet_bd_struct_t](#) *curBd)
Gets the control and the status region of the receive buffer descriptors.
- static uint16_t [ENET_HAL_GetTxBuffDescripControl](#) (volatile [enet_bd_struct_t](#) *curBd)
Gets the control and the status region of the transmit buffer descriptors.
- static uint16_t [ENET_HAL_GetRxbuffDescripExtControlOne](#) (volatile [enet_bd_struct_t](#) *curBd)
Gets the extended control region one of the receive buffer descriptor.
- static uint16_t [ENET_HAL_GetRxbuffDescripExtControlTwo](#) (volatile [enet_bd_struct_t](#) *curBd)

ENET HAL driver

- Gets the extended control region two of the receive buffer descriptor.
• static uint16_t **ENET_HAL_GetTxBuffDescripExtControl** (volatile **enet_bd_struct_t** *curBd)
 Gets the extended control region of the transmit buffer descriptors.
- static bool **ENET_HAL_GetTxBuffDescripTsFlag** (volatile **enet_bd_struct_t** *curBd)
 Gets the transmit buffer descriptor timestamp flag.
- uint16_t **ENET_HAL_GetBuffDescripLen** (volatile **enet_bd_struct_t** *curBd)
 Gets the data length of the buffer descriptors.
- uint8_t * **ENET_HAL_GetBuffDescripData** (volatile **enet_bd_struct_t** *curBd)
 Gets the buffer address of the buffer descriptors.
- uint32_t **ENET_HAL_GetBuffDescripTs** (volatile **enet_bd_struct_t** *curBd)
 Gets the timestamp of the buffer descriptors.
- static void **ENET_HAL_SetRxBuffDescripActive** (uint32_t baseAddr)
 Activates the receive buffer descriptor.
- static void **ENET_HAL_SetTxBuffDescripActive** (uint32_t baseAddr)
 Activates the transmit buffer descriptor.
- void **ENET_HAL_SetRMIIMode** (uint32_t baseAddr, **enet_config_rmii_t** *rmiiCfgPtr)
 Configures the (R)MII data interface of ENET.
- static void **ENET_HAL_SetRMIIISpeed** (uint32_t baseAddr, **enet_config_speed_t** speed)
 Configures the (R)MII speed of ENET.
- static void **ENET_HAL_SetSMI** (uint32_t baseAddr, uint32_t miiSpeed, uint32_t clkCycle, bool isPreambleDisabled)
 Configures the SMI(serial Management interface) of ENET.
- static bool **ENET_HAL_GetSMI** (uint32_t baseAddr)
 Gets SMI configuration status.
- static uint32_t **ENET_HAL_GetSMIData** (uint32_t baseAddr)
 Reads data from PHY.
- void **ENET_HAL_SetSMIRead** (uint32_t baseAddr, uint32_t phyAddr, uint32_t phyReg, **enet_mii_read_t** operation)
 Sets the SMI(serial Management interface) read command.
- void **ENET_HAL_SetSMIWrite** (uint32_t baseAddr, uint32_t phyAddr, uint32_t phyReg, **enet_mii_write_t** operation, uint32_t data)
 Sets the SMI(serial Management interface) write command.
- static void **ENET_HAL_Enable** (uint32_t baseAddr)
 Enables the ENET module.
- static void **ENET_HAL_Disable** (uint32_t baseAddr)
 Disables the ENET module.
- static void **ENET_HAL_SetEnhancedMacCmd** (uint32_t baseAddr, bool enable)
 Enables or disables the enhanced functionality of the MAC(1588 feature).
- void **ENET_HAL_SetIntMode** (uint32_t baseAddr, **enet_interrupt_request_t** source, bool enable)
 Enables/Disables the ENET interrupt.
- static void **ENET_HAL_ClearIntStatusFlag** (uint32_t baseAddr, **enet_interrupt_request_t** source)
 Clears ENET interrupt events.
- static bool **ENET_HAL_GetIntStatusFlag** (uint32_t baseAddr, **enet_interrupt_request_t** source)
 Gets the ENET interrupt status.
- static void **ENET_HAL_SetPromiscuousCmd** (uint32_t baseAddr, bool enable)
 Enables/disables the ENET promiscuous mode.
- static void **ENET_HAL_SetMibClearCmd** (uint32_t baseAddr, bool enable)
 Enables/disables the clear MIB counter.
- static void **ENET_HAL_SetMibCmd** (uint32_t baseAddr, bool enable)
 Sets the enable/disable of the MIB block.
- static bool **ENET_HAL_GetMibStatus** (uint32_t baseAddr)

- static void [ENET_HAL_SetTxAccelerator](#) (uint32_t baseAddr, uint32_t txConfig)

Gets the MIB idle status.

Sets the transmit accelerator.
- static void [ENET_HAL_SetRxAccelerator](#) (uint32_t baseAddr, uint32_t rxConfig)

Sets the receive accelerator.
- void [ENET_HAL_Set1588TimerStart](#) (uint32_t baseAddr, [enet_config_ptp_timer_t](#) *ptpCfgPtr)

Configures the 1588 timer and run the 1588 timer.
- void [ENET_HAL_Set1588TimerChnCmp](#) (uint32_t baseAddr, [enet_timer_channel_t](#) channel, uint32_t cmpValOld, uint32_t cmpValNew)

Configures the 1588 timer channel compare feature.
- void [ENET_HAL_Set1588Timer](#) (uint32_t baseAddr, [enet_config_ptp_timer_t](#) *ptpCfgPtr)

Initializes the 1588 timer.
- static void [ENET_HAL_Set1588TimerCmd](#) (uint32_t baseAddr, uint32_t enable)

Enables or disables the 1588 PTP timer.
- static void [ENET_HAL_Set1588TimerRestart](#) (uint32_t baseAddr)

Restarts the 1588 timer.
- static void [ENET_HAL_Set1588TimerAdjust](#) (uint32_t baseAddr, uint32_t increaseCorrection, uint32_t periodCorrection)

Adjusts the 1588 timer.
- static void [ENET_HAL_Set1588TimerCapture](#) (uint32_t baseAddr)

Sets the capture command to the 1588 timer.
- static void [ENET_HAL_Set1588TimerNewTime](#) (uint32_t baseAddr, uint32_t nanSecond)

Sets the 1588 timer.
- static uint32_t [ENET_HAL_Get1588TimerCurrentTime](#) (uint32_t baseAddr)

Gets the time from the 1588 timer.
- static void [ENET_HAL_Set1588TimerChnMode](#) (uint32_t baseAddr, [enet_timer_channel_t](#) channel, [enet_timer_channel_mode_t](#) mode)

Initializes one channel for the four-channel 1588 timer.
- static void [ENET_HAL_Set1588TimerChnInt](#) (uint32_t baseAddr, [enet_timer_channel_t](#) channel, bool enable)

Sets the 1588 time channel interrupt.
- static void [ENET_HAL_Set1588TimerChnCmpVal](#) (uint32_t baseAddr, [enet_timer_channel_t](#) channel, uint32_t compareValue)

Sets the compare value for the 1588 timer channel.
- static bool [ENET_HAL_Get1588TimerChnStatus](#) (uint32_t baseAddr, [enet_timer_channel_t](#) channel)

Gets the 1588 timer channel status.
- static void [ENET_HAL_Clear1588TimerChnFlag](#) (uint32_t baseAddr, [enet_timer_channel_t](#) channel)

Clears the 1588 timer channel interrupt flag.
- static uint32_t [ENET_HAL_GetTxTs](#) (uint32_t baseAddr)

Gets the transmit timestamp.
- static uint16_t [ENET_HAL_GetTxPackets](#) (uint32_t baseAddr)

Gets the transmit packet count statistic.
- static uint16_t [ENET_HAL_GetTxBroadCastPacket](#) (uint32_t baseAddr)

Gets the transmit broadcast packet statistic.
- static uint16_t [ENET_HAL_GetTxMultiCastPacket](#) (uint32_t baseAddr)

Gets the transmit multicast packet statistic.
- static uint16_t [ENET_HAL_GetTxCrcAlignErrorPacket](#) (uint32_t baseAddr)

Gets the transmit packets with CRC/Align error.

ENET HAL driver

- static uint16_t [ENET_HAL_GetTxUnderSizePacket](#) (uint32_t baseAddr)
Gets the transmit packets less than 64 bytes and good CRC.
- static uint16_t [ENET_HAL_GetTxFragPacket](#) (uint32_t baseAddr)
Gets the transmit packets less than 64 bytes and bad CRC.
- static uint16_t [ENET_HAL_GetTxOverSizePacket](#) (uint32_t baseAddr)
Gets the transmit packets over than MAX_FL bytes and good CRC.
- static uint16_t [ENET_HAL_GetTxJabPacket](#) (uint32_t baseAddr)
Gets the transmit packets over than MAX_FL bytes and bad CRC.
- static uint16_t [ENET_HAL_GetTxCollisionPacket](#) (uint32_t baseAddr)
Gets the transmit collision packets.
- static uint16_t [ENET_HAL_GetTxByte64Packet](#) (uint32_t baseAddr)
Gets the transmit 64-byte packet statistic.
- static uint16_t [ENET_HAL_GetTxByte65to127Packet](#) (uint32_t baseAddr)
Gets the transmit 65-byte to 127-byte packet statistic.
- static uint16_t [ENET_HAL_GetTxByte128to255Packet](#) (uint32_t baseAddr)
Gets the transmit packets 128-byte to 255-byte.
- static uint16_t [ENET_HAL_GetTxByte256to511Packet](#) (uint32_t baseAddr)
Gets the transmit packets 256-byte to 511-byte.
- static uint16_t [ENET_HAL_GetTxByte512to1023Packet](#) (uint32_t baseAddr)
Gets the transmit packets 512-byte to 1023-byte.
- static uint16_t [ENET_HAL_GetTxByte1024to2047Packet](#) (uint32_t baseAddr)
Gets the transmit packets 1024-byte to 2047-byte.
- static uint16_t [ENET_HAL_GetTxOverByte2048Packet](#) (uint32_t baseAddr)
Gets the transmit packets greater than 2048-byte.
- static uint32_t [ENET_HAL_GetTxOctets](#) (uint32_t baseAddr)
Gets the transmit octets.
- static uint16_t [ENET_HAL_GetTxFramesOk](#) (uint32_t baseAddr)
Gets the Frames transmitted OK.
- static uint16_t [ENET_HAL_GetTxFramesOneCollision](#) (uint32_t baseAddr)
Gets the Frames transmitted with single collision.
- static uint16_t [ENET_HAL_GetTxFramesMultiCollision](#) (uint32_t baseAddr)
Gets the frames transmitted with multiple collision.
- static uint16_t [ENET_HAL_GetTxFrameCarrSenseError](#) (uint32_t baseAddr)
Gets the frames transmitted with carrier sense error.
- static uint16_t [ENET_HAL_GetTxFramesDelay](#) (uint32_t baseAddr)
Gets the frames transmitted after deferral delay.
- static uint16_t [ENET_HAL_GetTxFramesLateCollision](#) (uint32_t baseAddr)
Gets the frames transmitted with late collision.
- static uint16_t [ENET_HAL_GetTxFramesExcessiveCollision](#) (uint32_t baseAddr)
Gets the frames transmitted with excessive collisions.
- static uint16_t [ENET_HAL_GetTxFramesMacError](#) (uint32_t baseAddr)
Gets the frames transmitted with the Tx FIFO underrun.
- static uint16_t [ENET_HAL_GetTxFramesPause](#) (uint32_t baseAddr)
Gets the transmitted flow control Pause Frames.
- static uint32_t [ENET_HAL_GetTxOctetFramesOk](#) (uint32_t baseAddr)
Gets the octet count for frames transmitted without error.
- static uint16_t [ENET_HAL_GetRxPackets](#) (uint32_t baseAddr)
Gets the receive packet count.
- static uint16_t [ENET_HAL_GetRxBroadCastPacket](#) (uint32_t baseAddr)
Gets the receive broadcast packet count.
- static uint16_t [ENET_HAL_GetRxMultiCastPacket](#) (uint32_t baseAddr)

- Gets the receive multicast packet count.
- static uint16_t [ENET_HAL_GetRxCrcAlignErrorPacket](#) (uint32_t baseAddr)
Gets the receive packets with CRC/Align error.
- static uint16_t [ENET_HAL_GetRxUnderSizePacket](#) (uint32_t baseAddr)
Gets the receive packets less than 64-byte and good CRC.
- static uint16_t [ENET_HAL_GetRxOverSizePacket](#) (uint32_t baseAddr)
Gets the receive packets greater than MAX_FL and good CRC.
- static uint16_t [ENET_HAL_GetRxFragPacket](#) (uint32_t baseAddr)
Gets the receive packets less than 64-byte and bad CRC.
- static uint16_t [ENET_HAL_GetRxJabPacket](#) (uint32_t baseAddr)
Gets the receive packets greater than MAX_FL and bad CRC.
- static uint16_t [ENET_HAL_GetRxByte64Packet](#) (uint32_t baseAddr)
Gets the receive packets with 64-byte.
- static uint16_t [ENET_HAL_GetRxByte65to127Packet](#) (uint32_t baseAddr)
Gets the receive packets with 65-byte to 127-byte.
- static uint16_t [ENET_HAL_GetRxByte128to255Packet](#) (uint32_t baseAddr)
Gets the receive packets with 128-byte to 255-byte.
- static uint16_t [ENET_HAL_GetRxByte256to511Packet](#) (uint32_t baseAddr)
Gets the receive packets with 256-byte to 511-byte.
- static uint16_t [ENET_HAL_GetRxByte512to1023Packet](#) (uint32_t baseAddr)
Gets the receive packets with 512-byte to 1023-byte.
- static uint16_t [ENET_HAL_GetRxByte1024to2047Packet](#) (uint32_t baseAddr)
Gets the receive packets with 1024-byte to 2047-byte.
- static uint16_t [ENET_HAL_GetRxOverByte2048Packet](#) (uint32_t baseAddr)
Gets the receive packets greater than 2048-byte.
- static uint32_t [ENET_HAL_GetRxOctets](#) (uint32_t baseAddr)
Gets the receive octets.
- static uint16_t [ENET_HAL_GetRxFramesDrop](#) (uint32_t baseAddr)
Gets the receive Frames not counted correctly.
- static uint16_t [ENET_HAL_GetRxFramesOk](#) (uint32_t baseAddr)
Gets the Frames received OK.
- static uint16_t [ENET_HAL_GetRxFramesCrcError](#) (uint32_t baseAddr)
Gets the Frames received with CRC error.
- static uint16_t [ENET_HAL_GetRxFramesAlignError](#) (uint32_t baseAddr)
Gets the Frames received with Alignment error.
- static uint16_t [ENET_HAL_GetRxFramesMacError](#) (uint32_t baseAddr)
Gets the FIFO overflow count.
- static uint16_t [ENET_HAL_GetRxFramesFlowControl](#) (uint32_t baseAddr)
Gets the received flow control Pause frames.
- static uint32_t [ENET_HAL_GetRxOctetsFramesOk](#) (uint32_t baseAddr)
Gets the octet count for Frames received without Error.

11.2.2 Data Structure Documentation

11.2.2.1 struct enet_bd_struct_t

Data Fields

- uint16_t length

ENET HAL driver

- **uint16_t control**
Buffer descriptor control.
- **uint8_t *buffer**
Data buffer pointer.
- **uint16_t controlExtend0**
Extend buffer descriptor control0.
- **uint16_t controlExtend1**
Extend buffer descriptor control1.
- **uint16_t payloadCheckSum**
Internal payload checksum.
- **uint8_t headerLength**
Header length.
- **uint8_t protocolType**
Protocol type.
- **uint16_t controlExtend2**
Extend buffer descriptor control2.
- **uint32_t timestamp**
Timestamp.

11.2.2.2 struct enet_config_rmii_t

Data Fields

- **enet_config_rmii_mode_t mode**
RMII/MII mode.
- **enet_config_speed_t speed**
100M/10M Speed
- **enet_config_duplex_t duplex**
Full/Duplex mode.
- **bool isRxOnTxDisabled**
Disable rx and tx.
- **bool isLoopEnabled**
MII loop mode.

11.2.2.3 struct enet_config_ptp_timer_t

Data Fields

- **bool isSlaveEnabled**
Master or slave PTP timer.
- **uint32_t clockIncease**
Timer increase value each clock period.
- **uint32_t period**
Timer period for generate interrupt event.

11.2.2.4 struct enet_config_tx_fifo_t

Data Fields

- bool `isStoreForwardEnabled`
Transmit FIFO store and forward.
- `uint8_t txFifoWrite`
Transmit FIFO write.
- `uint8_t txEmpty`
Transmit FIFO chapter empty threshold, default zero.
- `uint8_t txAlmostEmpty`
Transmit FIFO chapter almost empty threshold, The minimum value of 4 should be set.
- `uint8_t txAlmostFull`
Transmit FIFO chapter almost full threshold, The minimum value of 6 is required a recommended value of at least 8 should be set.

11.2.2.4.0.28 Field Documentation

11.2.2.4.0.28.1 `uint8_t enet_config_tx_fifo_t::txFifoWrite`

This should be set when `isStoreForwardEnabled` is false. this field indicates the number of bytes in step of 64 bytes written to the Tx FiFO before transmission of a frame begins

11.2.2.5 struct enet_config_rx_fifo_t

Data Fields

- `uint8_t rxFull`
Receive FIFO chapter full threshold, default zero.
- `uint8_t rxAlmostFull`
Receive FIFO chapter almost full threshold, The minimum value of 4 should be set.
- `uint8_t rxEmpty`
Receive FIFO chapter empty threshold, default zero.
- `uint8_t rxAlmostEmpty`
Receive FIFO chapter almost empty threshold, The minimum value of 4 should be set.

11.2.2.6 struct enet_mib_rx_stat_t

Data Fields

- `uint16_t rxPackets`
Receive packets.
- `uint16_t rxBroadcastPackets`
Receive broadcast packets.
- `uint16_t rxMulticastPackets`
Receive multicast packets.
- `uint16_t rxCrcAlignErrorPackets`
Receive packets with crc/align error.
- `uint16_t rxUnderSizeGoodPackets`

ENET HAL driver

- *Receive packets undersize and good crc.*
 - `uint16_t rxUnderSizeBadPackets`
Receive packets undersize and bad crc.
 - `uint16_t rxOverSizeGoodPackets`
Receive packets oversize and good crc.
 - `uint16_t rxOverSizeBadPackets`
Receive packets oversize and bad crc.
 - `uint16_t rxByte64Packets`
Receive packets 64-byte.
 - `uint16_t rxByte65to127Packets`
Receive packets 65-byte to 127-byte.
 - `uint16_t rxByte128to255Packets`
Receive packets 128-byte to 255-byte.
 - `uint16_t rxByte256to511Packets`
Receive packets 256-byte to 511-byte.
 - `uint16_t rxByte512to1023Packets`
Receive packets 512-byte to 1023-byte.
 - `uint16_t rxByte1024to2047Packets`
Receive packets 1024-byte to 2047-byte.
 - `uint16_t rxByteOver2048Packets`
Receive packets over 2048-byte.
 - `uint32_t rxOctets`
Receive octets.
 - `uint32_t ieeeOctetsrxFrameOk`
Receive octets of received Frames ok.
 - `uint16_t ieeerxFrameDrop`
Receive Frames dropped.
 - `uint16_t ieeerxFrameOk`
Receive Frames ok.
 - `uint16_t ieeerxFrameCrcErr`
Receive Frames with crc error.
 - `uint16_t ieeetxFrameAlignErr`
Receive Frames with align error.
 - `uint16_t ieeetxFrameMacErr`
Receive Frames with mac error.
 - `uint16_t ieeetxFramePause`
Receive flow control pause frames.

11.2.2.7 struct enet_mib_tx_stat_t

Data Fields

- `uint16_t txPackets`
Transmit packets.
- `uint16_t txBroadcastPackets`
Transmit broadcast packets.
- `uint16_t txMulticastPackets`
Transmit multicast packets.
- `uint16_t txCrcAlignErrorPackets`
Transmit packets with crc/align error.

- `uint16_t txUnderSizeGoodPackets`
Transmit packets undersize and good crc.
- `uint16_t txUnderSizeBadPackets`
Transmit packets undersize and bad crc.
- `uint16_t txOverSizeGoodPackets`
Transmit packets oversize and good crc.
- `uint16_t txOverSizeBadPackets`
Transmit packets oversize and bad crc.
- `uint16_t txCollision`
Transmit packets with collision.
- `uint16_t txByte64Packets`
Transmit packets 64-byte.
- `uint16_t txByte65to127Packets`
Transmit packets 65-byte to 127-byte.
- `uint16_t txByte128to255Packets`
Transmit packets 128-byte to 255-byte.
- `uint16_t txByte256to511Packets`
Transmit packets 256-byte to 511-byte.
- `uint16_t txByte512to1023Packets`
Transmit packets 512-byte to 1023-byte.
- `uint16_t txByte1024to2047Packets`
Transmit packets 1024-byte to 2047-byte.
- `uint16_t txByteOver2048Packets`
Transmit packets over 2048-byte.
- `uint32_t txOctets`
Transmit octets.
- `uint32_t ieeeOctetstxFrameOk`
Transmit octets of transmitted frames ok.
- `uint16_t ieeetxFrameOk`
Transmit frames ok.
- `uint16_t ieeetxFrameOneCollision`
Transmit frames with single collision.
- `uint16_t ieeetxFrameMultiCollision`
Transmit frames with multicast collision.
- `uint16_t ieeetxFrameLateCollision`
Transmit frames with late collision.
- `uint16_t ieeetxFrmaeExcCollision`
Transmit frames with excessive collision.
- `uint16_t ieeetxFrameDelay`
Transmit frames after deferral delay.
- `uint16_t ieeetxFrameMacErr`
Transmit frames with MAC error.
- `uint16_t ieeetxFrameCarrSenseErr`
Transmit frames with carrier sense error.
- `uint16_t ieeetxFramePause`
Transmit flow control Pause frame.

ENET HAL driver

11.2.2.8 struct enet_special_maccfg_t

Data Fields

- `uint16_t rxMaxFrameLen`
Receive maximum frame length.
- `uint16_t rxTruncLen`
Receive truncate length, must be greater than or equal to maximum frame length.
- `uint16_t txInterPacketGap`
Transmit inter-packet-gap.

11.2.2.9 struct enet_mac_config_t

Data Fields

- `enet_mac_operate_mode_t macMode`
Mac Normal or sleep mode.
- `uint8_t * macAddr`
MAC hardware address.
- `enet_config_rmii_t * rmiiCfgPtr`
RMII configure mode.
- `enet_config_rx_fifo_t * rxFifoPtr`
Receive fifo configuration, if NULL default values will be used.
- `enet_config_tx_fifo_t * txFifoPtr`
Transmit fifo configuration, if NULL default values will be used.
- `uint8_t rxAccelerCfg`
Receive accelerator configure, should be set when kEnetTxAccelEnable is set.
- `uint8_t txAccelerCfg`
Transmit accelerator configure, should be set when kEnetRxAccelEnable is set.
- `uint16_t pauseDuration`
Pause duration, should be set when kEnetRxFlowControlEnable is set.
- `enet_special_maccfg_t * macSpecialCfg`
special configure for MAC to instead of default configure

11.2.2.9.0.29 Field Documentation

11.2.2.9.0.29.1 enet_config_rmii_t* enet_mac_config_t::rmiiCfgPtr

Mac control configure, it is recommended to use `enet_mac_control_flag_t` it is special control set for loop mode, sleep mode, crc forward/terminate etc

11.2.3 Macro Definition Documentation

11.2.3.1 #define SYSTEM_LITTLE_ENDIAN (1)

```
11.2.3.2 #define ENET_ALIGN( x, align ) ((unsigned int)((x) + ((align)-1)) & (unsigned int)(~(unsigned int)((align)- 1)))
```

11.2.4 Enumeration Type Documentation

11.2.4.1 enum enet_status_t

Enumerator

kStatus_ENET_InvalidInput Invalid ENET input parameter.
kStatus_ENET_InvalidDevice Invalid ENET device.
kStatus_ENET_InitTimeout ENET initialize timeout.
kStatus_ENET_MemoryAllocateFail Memory allocate failure.
kStatus_ENET_GetClockFreqFail Get clock frequency failure.
kStatus_ENET_Initialized ENET device already initialized.
kStatus_ENET_Open Open ENET device.
kStatus_ENET_Close Close ENET device.
kStatus_ENET_Layer2UnInitialized Layer2 PTP buffer queue uninitialized.
kStatus_ENET_Layer2OverLarge Layer2 packet length over large.
kStatus_ENET_Layer2BufferFull Layer2 packet buffer full.
kStatus_ENET_Layer2TypeError Layer2 packet error type.
kStatus_ENET_PttringBufferFull PTP ring buffer full.
kStatus_ENET_PttringBufferEmpty PTP ring buffer empty.
kStatus_ENET_SMIUninitialized SMI uninitialized.
kStatus_ENET_SMIVisitTimeout SMI visit timeout.
kStatus_ENET_RxbdInvalid Receive buffer descriptor invalid.
kStatus_ENET_RxbdEmpty Receive buffer descriptor empty.
kStatus_ENET_RxbdTrunc Receive buffer descriptor truncate.
kStatus_ENET_RxbdError Receive buffer descriptor error.
kStatus_ENET_RxBdFull Receive buffer descriptor full.
kStatus_ENET_SmallRxBuffSize Receive buffer size is so small.
kStatus_ENET_NoEnoughRxBuffers Small receive buffer size.
kStatus_ENET_LargeBufferFull Receive large buffer full.
kStatus_ENET_TxLarge Transmit large packet.
kStatus_ENET_Txbdfull Transmit buffer descriptor full.
kStatus_ENET_TxbdNull Transmit buffer descriptor Null.
kStatus_ENET_TxBufferNull Transmit data buffer Null.
kStatus_ENET_NoRxBufferLeft No more receive buffer left.
kStatus_ENET_UnknownCommand Invalid ENET PTP IOCTL command.
kStatus_ENET_TimeOut ENET Timeout.
kStatus_ENET_MulticastPointerNull Null multicast group pointer.

ENET HAL driver

kStatus_ENET_NoMulticastAddr No multicast group address.
kStatus_ENET_AlreadyAddedMulticast Have Already added to multicast group.
kStatus_ENET_PHYAutoDiscoverFail Failed to automatically discover PHY.

11.2.4.2 enum enet_rx_bd_control_status_t

Enumerator

kEnetRxBdEmpty Empty bit.
kEnetRxBdRxSoftOwner1 Receive software owner.
kEnetRxBdWrap Update buffer descriptor.
kEnetRxBdRxSoftOwner2 Receive software owner.
kEnetRxBdLast Last BD in the frame.
kEnetRxBdMiss Receive for promiscuous mode.
kEnetRxBdBroadCast Broadcast.
kEnetRxBdMultiCast Multicast.
kEnetRxBdLengthViolation Receive length violation.
kEnetRxBdNoOctet Receive non-octet aligned frame.
kEnetRxBdCrc Receive CRC error.
kEnetRxBdOverRun Receive FIFO overrun.
kEnetRxBdTrunc Frame is truncated.

11.2.4.3 enum enet_rx_bd_control_extend0_t

Enumerator

kEnetRxBdIpv4 Ipv4 frame.
kEnetRxBdIpv6 Ipv6 frame.
kEnetRxBdVlan VLAN.
kEnetRxBdProtocolChecksumErr Protocol checksum error.
kEnetRxBdIpHeaderChecksumErr IP header checksum error.

11.2.4.4 enum enet_rx_bd_control_extend1_t

Enumerator

kEnetRxBdIntrrupt BD interrupt.
kEnetRxBdUnicast Unicast frame.
kEnetRxBdCollision BD collision.
kEnetRxBdPhyErr PHY error.
kEnetRxBdMacErr Mac error.

11.2.4.5 enum enet_tx_bd_control_status_t

Enumerator

- kEnetTxBdReady* Ready bit.
- kEnetTxBdTxSoftOwner1* Transmit software owner.
- kEnetTxBdWrap* Wrap buffer descriptor.
- kEnetTxBdTxSoftOwner2* Transmit software owner.
- kEnetTxBdLast* Last BD in the frame.
- kEnetTxBdTransmitCrc* Receive for transmit CRC.

11.2.4.6 enum enet_tx_bd_control_extend0_t

Enumerator

- kEnetTxBdTxErr* Transmit error.
- kEnetTxBdTxUnderFlowErr* Underflow error.
- kEnetTxBdExcessCollisionErr* Excess collision error.
- kEnetTxBdTxFrameErr* Frame error.
- kEnetTxBdLateCollisionErr* Late collision error.
- kEnetTxBdOverFlowErr* Overflow error.
- kEnetTxTimestampErr* Timestamp error.

11.2.4.7 enum enet_tx_bd_control_extend1_t

Enumerator

- kEnetTxBdTxInterrupt* Transmit interrupt.
- kEnetTxBdTimeStamp* Transmit timestamp flag.

11.2.4.8 enum enet_constant_parameter_t

Enumerator

- kEnetMacAddrLen* ENET mac address length.
- kEnetHashValMask* ENET hash value mask.
- kEnetMinBuffSize* ENET minimum buffer size.
- kEnetMaxTimeout* ENET timeout.
- kEnetMdcFreq* MDC frequency.

ENET HAL driver

11.2.4.9 enum enet_fifo_configure_t

Enumerator

kEnetMinTxFifoAlmostFull ENET minimum transmit fifo almost full value.

kEnetMinFifoAlmostEmpty ENET minimum FIFO almost empty value.

kEnetDefaultTxFifoAlmostFull ENET default tranmit fifo almost full value.

11.2.4.10 enum enet_mac_operate_mode_t

Enumerator

kEnetMacNormalMode Normal operationg mode for ENET MAC.

kEnetMacSleepMode Sleep mode for ENET MAC.

11.2.4.11 enum enet_config_rmii_mode_t

Enumerator

kEnetCfgMii MII mode for data interface.

kEnetCfgRmii RMII mode for data interface.

11.2.4.12 enum enet_config_speed_t

Enumerator

kEnetCfgSpeed100M Speed 100 M mode.

kEnetCfgSpeed10M Speed 10 M mode.

11.2.4.13 enum enet_config_duplex_t

Enumerator

kEnetCfgHalfDuplex Half duplex mode.

kEnetCfgFullDuplex Full duplex mode.

11.2.4.14 enum enet_mii_write_t

Enumerator

kEnetWriteNoCompliant Write frame operation, but not MII compliant.

kEnetWriteValidFrame Write frame operation for a valid MII management frame.

11.2.4.15 enum enet_mii_read_t

Enumerator

kEnetReadValidFrame Read frame operation for a valid MII management frame.*kEnetReadNoCompliant* Read frame operation, but not MII compliant.**11.2.4.16 enum enet_special_address_filter_t**

Enumerator

kEnetSpecialAddressInit Initializes the special address filter.*kEnetSpecialAddressEnable* Enables the special address filter.*kEnetSpecialAddressDisable* Disables the special address filter.**11.2.4.17 enum enet_timer_channel_t**

Enumerator

kEnetTimerChannel1 1588 timer Channel 1*kEnetTimerChannel2* 1588 timer Channel 2*kEnetTimerChannel3* 1588 timer Channel 3*kEnetTimerChannel4* 1588 timer Channel 4**11.2.4.18 enum enet_timer_channel_mode_t**

Enumerator

kEnetChannelDisable Disable timer channel.*kEnetChannelRisingCapture* Input capture on rising edge.*kEnetChannelFallingCapture* Input capture on falling edge.*kEnetChannelBothCapture* Input capture on both edges.*kEnetChannelSoftCompare* Output compare software only.*kEnetChannelToggleCompare* Toggle output on compare.*kEnetChannelClearCompare* Clear output on compare.*kEnetChannelSetCompare* Set output on compare.*kEnetChannelClearCompareSetOverflow* Clear output on compare, set output on overflow.*kEnetChannelSetCompareClearOverflow* Set output on compare, clear output on overflow.*kEnetChannelPulseLowonCompare* Pulse output low on compare for one 1588 clock cycle.*kEnetChannelPulseHighonCompare* Pulse output high on compare for one 1588 clock cycle.

11.2.4.19 enum enet_interrupt_request_t

Enumerator

- kEnetBabrInterrupt* Babbling receive error interrupt source.
- kEnetBabiInterrupt* Babbling transmit error interrupt source.
- kEnetGraceStopInterrupt* Graceful stop complete interrupt source.
- kEnetTxFrameInterrupt* TX FRAME interrupt source.
- kEnetTxByteInterrupt* TX BYTE interrupt source.
- kEnetRxFrameInterrupt* RX FRAME interrupt source.
- kEnetRxByteInterrupt* RX BYTE interrupt source.
- kEnetMiiInterrupt* MII interrupt source.
- kEnetEBusERInterrupt* Ethernet bus error interrupt source.
- kEnetLateCollisionInterrupt* Late collision interrupt source.
- kEnetRetryLimitInterrupt* Collision Retry Limit interrupt source.
- kEnetUnderrunInterrupt* Transmit FIFO underrun interrupt source.
- kEnetPayloadRxInterrupt* Payload Receive interrupt source.
- kEnetWakeUpInterrupt* WAKEUP interrupt source.
- kEnetTsAvailInterrupt* TS AVAIL interrupt source.
- kEnetTsTimerInterrupt* TS WRAP interrupt source.
- kEnetAllInterrupt* All interrupt.

11.2.4.20 enum enet_irq_number_t

Enumerator

- kEnetTsTimerNumber* ENET ts_timer irq number.
- kEnetReceiveNumber* ENET receive irq number.
- kEnetTransmitNumber* ENET transmit irq number.
- kEnetMiiErrorNumber* ENET mii error irq number.

11.2.4.21 enum enet_frame_max_t

Enumerator

- kEnetNsecOneSec* NanoSecond in one second.
- kEnetMaxFrameSize* Maximum frame size.
- kEnetMaxFrameVlanSize* Maximum VLAN frame size.
- kEnetMaxFrameDataSize* Maximum frame data size.
- kEnetDefaultTruncLen* Default Truncate length.
- kEnetDefaultIpg* ENET default transmit inter packet gap.
- kEnetMaxValidTxIpg* Maximum valid transmit IPG.
- kEnetMinValidTxIpg* Minimum valid transmit IPG.
- kEnetMaxMdioHoldCycle* Maximum hold time clock cycle on MDIO Output.

kEnetMaxFrameBdNumbers Maximum buffer descriptor numbers of a frame.

kEnetFrameFcsLen FCS length.

kEnetEthernetHeadLen Ethernet Frame header length.

kEnetEthernetVlanHeadLen Ethernet Vlan frame header length.

11.2.4.22 enum enet_txaccelerator_config_t

Enumerator

kEnetTxAccelIsShift16Enabled Tx FIFO shift-16.

kEnetTxAccelIpCheckEnabled Insert IP header checksum.

kEnetTxAccelProtoCheckEnabled Insert protocol checksum.

11.2.4.23 enum enet_rxaccelerator_config_t

Enumerator

kEnetRxAccelPadRemoveEnabled Padding removal for short IP frames.

kEnetRxAccelIpCheckEnabled Discard with wrong IP header checksum.

kEnetRxAccelProtoCheckEnabled Discard with wrong protocol checksum.

kEnetRxAccelMacCheckEnabled Discard with Mac layer errors.

kEnetRxAccelIsShift16Enabled Rx FIFO shift-16.

11.2.4.24 enum enet_mac_control_flag_t

Enumerator

kEnetStopModeEnable ENET Stop mode enable.

kEnetPayloadlenCheckEnable Enable MAC to enter hardware freeze when enter Debug mode. ENET receive payload length check Enable

kEnetRxFlowControlEnable Enable ENET flow control.

kEnetRxCrcFwdEnable Received frame crc is stripped from the frame.

kEnetRxPauseFwdEnable Pause frames are forwarded to the user application.

kEnetRxPadRemoveEnable Padding is removed from received frames.

kEnetRxBcRejectEnable Broadcast frame reject.

kEnetRxPromiscuousEnable Promiscuous mode enabled.

kEnetTxCrcFwdEnable Enable transmit frame with the crc from application.

kEnetTxCrcBdEnable When Tx CRC FWD disable, Tx buffer descriptor enable Transmit CRC.

kEnetMacAddrInsert Enable MAC address insert.

kEnetTxAccelEnable Transmit accelerator enable.

kEnetRxAccelEnable Transmit accelerator enable.

kEnetStoreAndFwdEnable Switcher to enable store and forward.

kEnetMacMibEnable Disable MIB module.

ENET HAL driver

kEnetSMIPreambleDisable Enable SMI preamble.

kEnetVlanTagEnabled Enable Vlan Tag.

kEnetMacEnhancedEnable Enable enhanced MAC feature (1588 feature/enhanced buff descriptor)

11.2.5 Function Documentation

11.2.5.1 **uint32_t ENET_HAL_Init (uint32_t *baseAddr*)**

Parameters

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

Returns

The status of the initialize operation.

- kStatus_ENET_InitTimeout initialize failure for timeout.
- kStatus_ENET_Success initialize success.

11.2.5.2 **void ENET_HAL_SetMac (uint32_t *baseAddr*, const enet_mac_config_t * *macCfgPtr*, uint32_t *sysClk*)**

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>macCfgPtr</i>	The mac configure structure.
<i>sysClk</i>	The system clock for ENET module.

11.2.5.3 **void ENET_HAL_SetTxBuffDescriptors (uint32_t *baseAddr*, volatile enet_bd_struct_t * *txBds*, uint8_t * *txBuffer*, uint32_t *txBdNumber*, uint32_t *txBuffSizeAlign*)**

Parameters

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

<i>txBds</i>	The start address of ENET transmit buffer descriptors. This address must always be evenly divisible by 16.
<i>txBuffer</i>	The transmit data buffer start address. This address must always be evenly divisible by 16.
<i>txBdNumber</i>	The transmit buffer descriptor numbers.
<i>txBuffSizeAlign</i>	The aligned transmit buffer size.

11.2.5.4 void ENET_HAL_SetRxBuffDescriptors (uint32_t *baseAddr*, volatile enet_bd_struct_t * *rxBds*, uint8_t * *rxBuffer*, uint32_t *rxBdNumber*, uint32_t *rxBuffSizeAlign*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>rxBds</i>	The start address of ENET receive buffer descriptors. This address must always be evenly divisible by 16.
<i>rxBuffer</i>	The receive data buffer start address. This address must always be evenly divisible by 16.
<i>rxBdNumber</i>	The receive buffer descriptor numbers.
<i>rxBuffSizeAlign</i>	The aligned receive transmit buffer size.

11.2.5.5 void ENET_HAL_SetFifo (uint32_t *baseAddr*, const enet_mac_config_t * *macCfgPtr*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>macCfgPtr</i>	The ENET MAC configuration structure.

11.2.5.6 void ENET_HAL_GetMibRxStat (uint32_t *baseAddr*, enet_mib_rx_stat_t * *rxStat*)

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>rxStat</i>	The received statistics from MIB.

11.2.5.7 void ENET_HAL_GetMibTxStat (uint32_t *baseAddr*, enet_mib_tx_stat_t * *txStat*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>txStat</i>	The received statistics from MIB.

11.2.5.8 static void ENET_HAL_SetStopCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This controls device behavior in doze mode.

In doze mode, all clocks of the ENET assembly are disabled, except the RMII/MII clock. Doze mode is similar to a conditional stop mode entry for the ENET assembly depending on the stop enable signal control enable/disable.

Parameters

<i>baseAddr</i>	The ENET peripheral base address..
<i>enable</i>	The switch to enable/disable stop mode. <ul style="list-style-type: none">• true to enable the stop mode.• false to disable the stop mode.

11.2.5.9 static void ENET_HAL_SetDebugCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Enabling the debug mode enables Mac to enter the hardware freeze when the device enters debug mode. Disabling the debug mode enables Mac to continue operation in debug mode.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	<p>The switch to enable/disable the sleep mode.</p> <ul style="list-style-type: none"> • true MAC enter hardware freeze mode in debug mode. • false MAC continues operation in debug mode.

11.2.5.10 void ENET_HAL_SetMacMode (uint32_t *baseAddr*, enet_mac_operate_mode_t *mode*)

Enabling the sleep mode disables the normal operating mode. When enabling the sleep mode, the magic packet detection is also enabled so that a remote agent can wake up the node.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>mode</i>	The normal operating mode or sleep mode.

11.2.5.11 void ENET_HAL_SetMacAddr (uint32_t *baseAddr*, uint8_t * *hwAddr*)

This interface sets the six-byte Mac address of the ENET interface.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>hwAddr</i>	The pointer to the array of the six-bytes Mac address. The six-bytes mac address is used by ENET MAC to filtering the incoming Ethernet frames.

11.2.5.12 static void ENET_HAL_SetMacAddrInsertCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This interface enables the six-byte Mac address modification on transmit. If this is enabled, the MAC overwrites the source Mac address with the given Mac address through the ENET_HAL_SetMacAddr.

Parameters

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

ENET HAL driver

<i>enable</i>	The switch to enable/disable the Mac address modification on transmit. <ul style="list-style-type: none">• true enable Mac address modification on transmit.• false disable Mac address modification on transmit.
---------------	--

11.2.5.13 void ENET_HAL_SetMulticastAddrHash (*uint32_t baseAddr, uint32_t crcValue, enet_special_address_filter_t mode*)

This interface is used to add the ENET device to a multicast group address. After joining the group, Mac receives all frames with the group Mac address.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>crcValue</i>	The CRC value of the multicast group address.
<i>mode</i>	The operation for init/enable/disable the specified hardware address.

11.2.5.14 void ENET_HAL_SetUnicastAddrHash (*uint32_t baseAddr, uint32_t crcValue, enet_special_address_filter_t mode*)

This interface is used to add an individual address to the hardware address filter. Mac receives all frames with the individual address as a destination address.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>crcValue</i>	The CRC value of the special address.
<i>mode</i>	The operation for init/enable/disable the specified hardware address.

11.2.5.15 static void ENET_HAL_SetTxcrcFwdCmd (*uint32_t baseAddr, bool enable*) [*inline*], [*static*]

If transmitting the CRC forward is enabled, the transmitter does not append a CRC to transmitted frames, because it is expecting a frame with the CRC from the application. If transmitting the CRC forward is disabled, the transmit buffer descriptor controls whether the frame has a CRC from the application.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	<p>The switch to enable/disable forwarding frame from application with CRC for transmitted frames.</p> <ul style="list-style-type: none"> • True the transmitter forwarding the frames with CRC from the application. so the transmitter doesn't append any CRC to transmitted frames. • False the transmit buffer descriptor controls whether the frame has a CRC from the application.

11.2.5.16 static void ENET_HAL_SetRxcrcFwdCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function decides whether the CRC field of the received frame is transmitted or stripped. Enabling this feature strips the CRC field from the frame. If padding remove is enabled, this feature is ignored and the CRC field is checked and always terminated and removed. Note that if the padding is enabled, the CRC field is checked and always terminated and removed.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	<p>The switch to enable/disable transmit the receive CRC.</p> <ul style="list-style-type: none"> • True to transmit the received CRC. • False to strip the received CRC.

11.2.5.17 static void ENET_HAL_SetPauseFwdCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This is used to decide whether pause frames is forwarded or discarded.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	<p>The switch to enable/disable forward PAUSE frames</p> <ul style="list-style-type: none"> • True to forward PAUSE frames. • False to terminate and discard PAUSE frames.

ENET HAL driver

**11.2.5.18 static void ENET_HAL_SetPadRemoveCmd (uint32_t *baseAddr*, bool *enable*)
[inline], [static]**

Enabling the frame padding remove removes the padding from the received frames.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable remove padding <ul style="list-style-type: none"> • True to remove padding from frames. • False to disable padding remove.

11.2.5.19 static void ENET_HAL_SetFlowControlCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

If the flow control is enabled, the receive detects paused frames. Upon pause frame detection, the transmitter stops transmitting data frames for a given duration.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable flow control. <ul style="list-style-type: none"> • True to enable the flow control. • False to disable the flow control.

11.2.5.20 static void ENET_HAL_SetBroadcastRejectCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

If the broadcast frame reject is enabled, frames with destination address equal to 0xffff_ffff are rejected unless the promiscuous mode is open.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable reject broadcast frames. <ul style="list-style-type: none"> • True to reject broadcast frames. • False to accept broadcast frames.

11.2.5.21 static void ENET_HAL_SetPayloadCheckCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

If the length/type is less than 0x600, when enable payload length check is enabled, the core checks the frame's payload length. If the length/type is greater than or equal to 0x600, Mac interprets the field as a

ENET HAL driver

type and no payload length check is performed.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable payload length check. <ul style="list-style-type: none"> • True to enable payload length check. • False to disable payload length check.

11.2.5.22 static void ENET_HAL_SetGraceTxStopCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

When this field is set, Mac stops transmission after a currently transmitted frame is complete.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable graceful transmit stop <ul style="list-style-type: none"> • True to enable graceful transmit stop. • False to disable graceful transmit stop.

11.2.5.23 static void ENET_HAL_SetPauseDuration (uint32_t *baseAddr*, uint32_t *pauseDuration*) [inline], [static]

This function sets the pause duration used in a transmission of a PAUSE frame. When another node detects a PAUSE frame, that node pauses transmission for the pause duration.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>pauseDuration</i>	The PAUSE duration for the transmitted PAUSE frame the maximum pause duration is 0xFFFF.

11.2.5.24 static void ENET_HAL_SetTxPauseCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function enables pausing the frame transmission. When this is set, with transmission of data frames stopped, Mac transmits a Mac control pause frame. Next, Mac clears and resumes transmitting data frames.

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable transmit pause frame.

11.2.5.25 static bool ENET_HAL_GetTxPause (uint32_t *baseAddr*) [inline], [static]

This function gets the transmitted pause frame status.

Parameters

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

Returns

The status of the received flow control frames.

- True if the MAC is transmitting a MAC control PAUSE frame.
- False if No PAUSE frame transmit.

11.2.5.26 static bool ENET_HAL_GetRxPause (uint32_t *baseAddr*) [inline], [static]

This function gets the received pause frame status.

Parameters

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

Returns

The status of the received flow control frames

- True if the flow control pause frame is received and the transmitter pauses for the duration defined in this pause frame.
- False if there is no flow control frame received or the pause duration is complete.

11.2.5.27 void ENET_HAL_SetTxInterPacketGap (uint32_t *baseAddr*, uint32_t *ipgValue*)

This function indicates the inter-packet gap, in bytes, between transmitted frames. Valid values range from 8 to 27. If value is less than 8, the IPG is 8. If value is greater than 27, the IPG is 27.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>ipgValue</i>	The Inter-Packet-Gap for transmitted frames The default value is 12, the maximum value set to IPG is 0x1F.

11.2.5.28 static void ENET_HAL_SetTruncLen (uint32_t *baseAddr*, uint32_t *length*) [inline], [static]

This function indicates the value a receive frame is truncated, if it is greater than this value. The frame truncation length must be greater than or equal to the receive maximum frame length.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>length</i>	The truncation length. The maximum value is 0x3FFF The default truncation length is 2047(0x7FF).

11.2.5.29 static void ENET_HAL_SetRxMaxFrameLen (uint32_t *baseAddr*, uint32_t *maxFrameSize*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>maxFrameSize</i>	The maximum receive frame size, the reset value is 1518 or 1522 if the VLAN tags are supported. The length is measured starting at DA and including the CRC.

11.2.5.30 static uint16_t ENET_HAL_GetRxMaxFrameLen (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The maximum receive frame size, The length is measured starting at DA and including the CRC.

ENET HAL driver

11.2.5.31 **static void ENET_HAL_SetRxMaxBuffSize (*uint32_t baseAddr, uint32_t maxBufferSize*) [inline], [static]**

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>maxBufferSize</i>	The maximum receive buffer size, which should not be smaller than 256 It should be evenly divisible by 16 and the maximum receive size should not be larger than 0x3ff0.

11.2.5.32 void ENET_HAL_SetTxFifo (uint32_t *baseAddr*, enet_config_tx_fifo_t * *thresholdCfg*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>thresholdCfg</i>	The FIFO threshold configuration

11.2.5.33 void ENET_HAL_SetRxFifo (uint32_t *baseAddr*, enet_config_rx_fifo_t * *thresholdCfg*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>thresholdCfg</i>	The FIFO threshold configuration

11.2.5.34 static void ENET_HAL_SetRxBuffDescripAddr (uint32_t *baseAddr*, uint32_t *rxBdAddr*) [inline], [static]

This interface provides the beginning of the receive buffer descriptor queue in the external memory. The rxBdAddr is recommended to be 128-bit aligned, must be evenly divisible by 16.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>rxBdAddr</i>	The start address of receive buffer descriptor. This address must always be evenly divisible by 16.

11.2.5.35 static void ENET_HAL_SetTxBuffDescripAddr (uint32_t *baseAddr*, uint32_t *txBdAddr*) [inline], [static]

This interface provides the beginning of the transmit buffer descriptor queue in the external memory. The txBdAddr is recommended to be 128-bit aligned, must be evenly divisible by 16.

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>txBdAddr</i>	The start address of transmit buffer descriptors This address must always be evenly divisible by 16.

11.2.5.36 void ENET_HAL_InitRxBuffDescriptors (volatile enet_bd_struct_t * rxBds, uint8_t * rxBuff, uint32_t rxbdNum, uint32_t rxBuffSizeAlign)

To make sure the uDMA will do the right data transfer after you activate with wrap flag and all the buffer descriptors should be initialized with an empty bit.

Parameters

<i>rxBds</i>	The current receive buffer descriptor.
<i>rxBuff</i>	The data buffer on receive buffer descriptor. This address must always be evenly divisible by 16.
<i>rxbdNum</i>	The number of the receive buffer descriptors.
<i>rxBuffSizeAlign</i>	The aligned receive buffer size.

11.2.5.37 void ENET_HAL_InitTxBuffDescriptors (volatile enet_bd_struct_t * txBds, uint8_t * txBuff, uint32_t txbdNum, uint32_t txBuffSizeAlign)

To make sure the uDMA will do the right data transfer after you active with wrap flag.

Parameters

<i>txBds</i>	The current transmit buffer descriptor.
<i>txBuff</i>	The data buffer on transmit buffer descriptor.
<i>txbdNum</i>	The number of transmit buffer descriptors.
<i>txBuffSizeAlign</i>	The aligned transmit buffer size.

11.2.5.38 void ENET_HAL_UpdateRxBuffDescriptor (volatile enet_bd_struct_t * rxBds, uint8_t * data, bool isbufferUpdate)

This interface mainly clears the status region and updates the received buffer descriptor to ensure that the BD is correctly used.

Parameters

<i>rxBds</i>	The current receive buffer descriptor.
<i>data</i>	The data buffer address. This address must be divided by 16 if the <i>isbufferUpdate</i> is set.
<i>isbufferUpdate</i>	The data buffer update flag. When you want to update the data buffer of the buffer descriptor ensure that this flag is set.

11.2.5.39 void ENET_HAL_UpdateTxBuffDescriptor (volatile enet_bd_struct_t * *txBds*, uint16_t *length*, bool *isTxtsCfged*, bool *isTxCrcEnable*, bool *isLastOne*)

This interface mainly clears the status region and updates the transmit buffer descriptor to ensure tat the BD is correctly used again. You should set the *isTxtsCfged* when the transmit timestamp feature is required.

Parameters

<i>txBds</i>	The current transmit buffer descriptor.
<i>length</i>	The data length on buffer descriptor.
<i>isTxtsCfged</i>	The timestamp configure flag. The timestamp is added to the transmit buffer descriptor when this flag is set.
<i>isTxCrcEnable</i>	<p>The flag to transmit CRC sequence after the data byte.</p> <ul style="list-style-type: none"> • True the transmit controller transmits the CRC sequence after the data byte. if the transmit CRC forward from application is disabled this flag should be set to add the CRC sequence. • False the transmit buffer descriptor does not transmit the CRC sequence after the data byte. if the transmit CRC forward from application.
<i>isLastOne</i>	<p>The last BD flag in a frame.</p> <ul style="list-style-type: none"> • True the last BD in a frame. • False not the last BD in a frame.

11.2.5.40 static void ENET_HAL_ClearTxBuffDescriptor (volatile enet_bd_struct_t * *curBd*) [inline], [static]

Clears the data, length, control, and status region of the transmit buffer descriptor.

ENET HAL driver

Parameters

<i>curBd</i>	The current buffer descriptor.
--------------	--------------------------------

11.2.5.41 static uint16_t ENET_HAL_GetRxBuffDescripControl (volatile enet_bd_struct_t * *curBd*) [inline], [static]

This interface can get the whole control and status region of the receive buffer descriptor. The enet_rx_bd_control_status_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

<i>curBd</i>	The current receive buffer descriptor.
--------------	--

Returns

The control and status data on buffer descriptors.

11.2.5.42 static uint16_t ENET_HAL_GetTxBuffDescripControl (volatile enet_bd_struct_t * *curBd*) [inline], [static]

This interface can get the whole control and status region of the transmit buffer descriptor. The enet_tx_bd_control_status_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

<i>curBd</i>	The current transmit buffer descriptor.
--------------	---

Returns

The extended control region of transmit buffer descriptor.

11.2.5.43 static uint16_t ENET_HAL_GetRxbuffDescripExtControlOne (volatile enet_bd_struct_t * *curBd*) [inline], [static]

This interface can get the whole control and status region of the receive buffer descriptor. The enet_rx_bd_control_extend0_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

<i>curBd</i>	The current receive buffer descriptor.
--------------	--

Returns

The extended control region0 data of receive buffer descriptor.

11.2.5.44 static uint16_t ENET_HAL_GetRxbuffDescripExtControlTwo (volatile enet_bd_struct_t * *curBd*) [inline], [static]

This interface can get the whole control and status region of the receive buffer descriptor. The enet_rx_bd_control_extend1_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

<i>curBd</i>	The current receive buffer descriptor.
--------------	--

Returns

The extended control region1 data of receive buffer descriptor.

11.2.5.45 static uint16_t ENET_HAL_GetTxBuffDescripExtControl (volatile enet_bd_struct_t * *curBd*) [inline], [static]

This interface can get the whole control and status region of the transmit buffer descriptor. The enet_tx_bd_control_extend0_t enum type definition should be used if you want to get each status bit of the control and status region.

Parameters

<i>curBd</i>	The current transmit buffer descriptor.
--------------	---

Returns

The extended control data.

11.2.5.46 static bool ENET_HAL_GetTxBuffDescripTsFlag (volatile enet_bd_struct_t * *curBd*) [inline], [static]

ENET HAL driver

Parameters

<i>curBd</i>	The ENET transmit buffer descriptor.
--------------	--------------------------------------

Returns

true if timestamp region is set else false.

11.2.5.47 uint16_t ENET_HAL_GetBuffDescripLen (volatile enet_bd_struct_t * *curBd*)

Parameters

<i>curBd</i>	The current buffer descriptor.
--------------	--------------------------------

Returns

The data length of the buffer descriptor.

11.2.5.48 uint8_t* ENET_HAL_GetBuffDescripData (volatile enet_bd_struct_t * *curBd*)

Parameters

<i>curBd</i>	The current buffer descriptor.
--------------	--------------------------------

Returns

The buffer address of the buffer descriptor.

11.2.5.49 uint32_t ENET_HAL_GetBuffDescripTs (volatile enet_bd_struct_t * *curBd*)

Parameters

<i>curBd</i>	The current buffer descriptor.
--------------	--------------------------------

Returns

The time stamp of the frame in the buffer descriptor. Notice that the frame timestamp is only set in the last buffer descriptor of the frame.

**11.2.5.50 static void ENET_HAL_SetRxBuffDescripActive (uint32_t *baseAddr*)
[inline], [static]**

The buffer descriptor activation should be done after the ENET module is enabled. Otherwise, the activation fails.

ENET HAL driver

Parameters

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

11.2.5.51 static void ENET_HAL_SetTxBuffDescipActive (uint32_t *baseAddr*) [inline], [static]

The buffer descriptor activation should be done after the ENET module is enabled. Otherwise, the activation fails.

Parameters

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

11.2.5.52 void ENET_HAL_SetRMIIIMode (uint32_t *baseAddr*, enet_config_rmii_t * *rmiiCfgPtr*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>rmiiCfgPtr</i>	The RMII/MII configuration structure pointer.

11.2.5.53 static void ENET_HAL_SetRMIIISpeed (uint32_t *baseAddr*, enet_config_speed_t *speed*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>speed</i>	The RMII/MII speed.

11.2.5.54 static void ENET_HAL_SetSMI (uint32_t *baseAddr*, uint32_t *miiSpeed*, uint32_t *clkCycle*, bool *isPreambleDisabled*) [inline], [static]

Sets the SMI(MDC/MDIO) between Mac and PHY. The miiSpeed is a value that controls the frequency of the MDC, relative to the internal module clock(InterClockSrc). A value of zero in this parameter turns the MDC off and leaves it in the low voltage state. Any non-zero value results in the MDC frequency $MDC = \text{InterClockSrc}/((\text{miiSpeed} + 1)*2)$. So $\text{miiSpeed} = \text{InterClockSrc}/(2*\text{MDC}) - 1$. The Maximum MDC clock is 2.5MHZ(maximum). The recommended action is to round up and plus one to simplify: $\text{miiSpeed} = \text{InterClockSrc}/(2*2.5\text{MHZ})$.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>miiSpeed</i>	The MII speed and it is ranged from 0~0x3F.
<i>clkCycle</i>	The hold on clock cycles for MDIO output
<i>isPreamble-Disabled</i>	The preamble disabled flag.

11.2.5.55 static bool ENET_HAL_GetSMI(uint32_t *baseAddr*) [inline], [static]

This interface is usually called to check the SMI(serial Management interface) before the Mac writes or reads the PHY registers.

Parameters

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

Returns

The MII configuration status.

- true if the MII has been configured.
- false if the MII has not been configured.

11.2.5.56 static uint32_t ENET_HAL_GetSMIData(uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The data read from PHY

11.2.5.57 void ENET_HAL_SetSMIRead(uint32_t *baseAddr*, uint32_t *phyAddr*, uint32_t *phyReg*, enet_mii_read_t *operation*)

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register.
<i>operation</i>	The read operation.

11.2.5.58 void ENET_HAL_SetSMIWrite (uint32_t *baseAddr*, uint32_t *phyAddr*, uint32_t *phyReg*, enet_mii_write_t *operation*, uint32_t *data*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>phyAddr</i>	The PHY address.
<i>phyReg</i>	The PHY register.
<i>operation</i>	The write operation.
<i>data</i>	The data written to PHY.

11.2.5.59 static void ENET_HAL_Enable (uint32_t *baseAddr*) [inline], [static]

Parameters

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

11.2.5.60 static void ENET_HAL_Disable (uint32_t *baseAddr*) [inline], [static]

Parameters

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

11.2.5.61 static void ENET_HAL_SetEnhancedMacCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The enable/disable switch to the enhanced functionality of the MAC(1588 feature).

11.2.5.62 void ENET_HAL_SetIntMode (*uint32_t baseAddr*, *enet_interrupt_request_t source*, *bool enable*)

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>source</i>	The interrupt sources.
<i>enable</i>	The interrupt enable switch.

11.2.5.63 static void ENET_HAL_ClearIntStatusFlag (*uint32_t baseAddr*, *enet_interrupt_request_t source*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>source</i>	The interrupt source to be cleared. <i>enet_interrupt_request_t</i> enum types is recommended as the interrupt source.

11.2.5.64 static bool ENET_HAL_GetIntStatusFlag (*uint32_t baseAddr*, *enet_interrupt_request_t source*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>source</i>	The interrupt sources. <i>enet_interrupt_request_t</i> enum types is recommended as the interrupt source.

Returns

- The event status of the interrupt source
 - true if the interrupt event happened.
 - false if the interrupt event has not happened.

11.2.5.65 static void ENET_HAL_SetPromiscuousCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable the promiscuous mode.

**11.2.5.66 static void ENET_HAL_SetMibClearCmd (uint32_t *baseAddr*, bool *enable*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable the MIB counter.

**11.2.5.67 static void ENET_HAL_SetMibCmd (uint32_t *baseAddr*, bool *enable*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The switch to enable/disable the MIB block. <ul style="list-style-type: none"> • True to enable MIB block. • False to disable MIB block.

**11.2.5.68 static bool ENET_HAL_GetMibStatus (uint32_t *baseAddr*) [inline],
[static]**

Parameters

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

Returns

true if in MIB idle and MIB is not updating else false.

11.2.5.69 static void ENET_HAL_SetTxAccelerator (uint32_t *baseAddr*, uint32_t *txConfig*) [inline], [static]

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>txConfig</i>	The transmit accelerator configuration. The enet_txaccelerator_config_t is recommended to be used to do the configure.

11.2.5.70 static void ENET_HAL_SetRxAccelerator (uint32_t *baseAddr*, uint32_t *rxConfig*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>rxConfig</i>	The receive accelerator configuration. The enet_rxaccelerator_config_t is recommended to be used to do the configure.

11.2.5.71 void ENET_HAL_Set1588TimerStart (uint32_t *baseAddr*, enet_config_ptp_timer_t * *ptpCfgPtr*)

This interface configures the 1588 timer and starts the 1588 timer. After the timer starts the 1588 timer starts incrementing.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>ptpCfgPtr</i>	The 1588 timer configuration structure pointer.

11.2.5.72 void ENET_HAL_Set1588TimerChnCmp (uint32_t *baseAddr*, enet_timer_channel_t *channel*, uint32_t *cmpValOld*, uint32_t *cmpValNew*)

This interface configures the 1588 timer channel with the compare feature.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>channel</i>	The 1588 timer channel.

<i>cmpValOld</i>	The old compare value.
<i>cmpValNew</i>	The new compare value.

11.2.5.73 void ENET_HAL_Set1588Timer (uint32_t *baseAddr*, enet_config_ptp_timer_t * *ptpCfgPtr*)

This interface initializes the 1588 context structure. Initialize 1588 parameters according to the user configuration structure.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>ptpCfgPtr</i>	The 1588 timer configuration.

11.2.5.74 static void ENET_HAL_Set1588TimerCmd (uint32_t *baseAddr*, uint32_t *enable*) [inline], [static]

Enable the PTP timer will starts the timer. Disable the timer will stop timer at the current value.

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>enable</i>	The 1588 timer Enable switch <ul style="list-style-type: none"> • True enabled the 1588 PTP timer. • False disable or stop the 1588 PTP timer.

11.2.5.75 static void ENET_HAL_Set1588TimerRestart (uint32_t *baseAddr*) [inline], [static]

Restarting the PTP timer clears all PTP-timer counters to zero.

Parameters

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

11.2.5.76 static void ENET_HAL_Set1588TimerAdjust (uint32_t *baseAddr*, uint32_t *increaseCorrection*, uint32_t *periodCorrection*) [inline], [static]

Adjust the 1588 timer according to the increase and correction period of the configured correction.

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>increase-Correction</i>	The increase correction for 1588 timer.
<i>period-Correction</i>	The period correction for 1588 timer.

11.2.5.77 static void ENET_HAL_Set1588TimerCapture (uint32_t *baseAddr*) [inline], [static]

This is used before reading the current time register. After set timer capture, please wait for about 1us before read the captured timer.

Parameters

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

11.2.5.78 static void ENET_HAL_Set1588TimerNewTime (uint32_t *baseAddr*, uint32_t *nanSecond*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>nanSecond</i>	The nanosecond set to 1588 timer.

11.2.5.79 static uint32_t ENET_HAL_Get1588TimerCurrentTime (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

the current time from 1588 timer.

11.2.5.80 **static void ENET_HAL_Set1588TimerChnMode (uint32_t *baseAddr*,
enet_timer_channel_t *channel*, enet_timer_channel_mode_t *mode*)**
[**inline**], [**static**]

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>channel</i>	The 1588 timer channel number.
<i>mode</i>	Compare or capture mode for the four-channel 1588 timer channel.

11.2.5.81 static void ENET_HAL_Set1588TimerChnInt (uint32_t *baseAddr*, enet_timer_channel_t *channel*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>channel</i>	The 1588 timer channel number.
<i>enable</i>	The switch to enable or disable interrupt.

11.2.5.82 static void ENET_HAL_Set1588TimerChnCmpVal (uint32_t *baseAddr*, enet_timer_channel_t *channel*, uint32_t *compareValue*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>channel</i>	The 1588 timer channel number.
<i>compareValue</i>	Compare value for 1588 timer channel.

11.2.5.83 static bool ENET_HAL_Get1588TimerChnStatus (uint32_t *baseAddr*, enet_timer_channel_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>channel</i>	The 1588 timer channel number.

Returns

Compare or capture operation status

- True if the compare or capture has occurred.
- False if the compare or capture has not occurred.

11.2.5.84 **static void ENET_HAL_Clear1588TimerChnFlag (uint32_t *baseAddr*, enet_timer_channel_t *channel*) [inline], [static]**

ENET HAL driver

Parameters

<i>baseAddr</i>	The ENET peripheral base address.
<i>channel</i>	The 1588 timer channel number.

11.2.5.85 static uint32_t ENET_HAL_GetTxTs (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The timestamp of the last transmitted frame.

11.2.5.86 static uint16_t ENET_HAL_GetTxPackets (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packet count.

11.2.5.87 static uint16_t ENET_HAL_GetTxBroadCastPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit broadcast packet statistic.

11.2.5.88 static uint16_t ENET_HAL_GetTxMultiCastPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit multicast packet statistic.

11.2.5.89 static uint16_t ENET_HAL_GetTxCrcAlignErrorPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packets with CRC/Align error.

11.2.5.90 static uint16_t ENET_HAL_GetTxUnderSizePacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packets less than 64 bytes and good CRC.

11.2.5.91 static uint16_t ENET_HAL_GetTxFragPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

ENET HAL driver

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

Returns

The transmit packets less than 64 bytes and bad CRC.

11.2.5.92 static uint16_t ENET_HAL_GetTxOverSizePacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packets over size than MAX_FL and good CRC.

11.2.5.93 static uint16_t ENET_HAL_GetTxJabPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packets over size than MAX_FL bytes and bad CRC.

11.2.5.94 static uint16_t ENET_HAL_GetTxCollisionPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit collision packets.

11.2.5.95 static uint16_t ENET_HAL_GetTxByte64Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit 64-byte packet.

11.2.5.96 static uint16_t ENET_HAL_GetTxByte65to127Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit 65-byte to 127-byte packet statistic.

11.2.5.97 static uint16_t ENET_HAL_GetTxByte128to255Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packets 128 byte to 255-byte.

11.2.5.98 static uint16_t ENET_HAL_GetTxByte256to511Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The transmit packets 256 byte to 511-byte.

**11.2.5.99 static uint16_t ENET_HAL_GetTxByte512to1023Packet (uint32_t *baseAddr*)
[inline], [static]**

Parameters

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

Returns

The transmit packets 512 byte to 1023-byte.

**11.2.5.100 static uint16_t ENET_HAL_GetTxByte1024to2047Packet (uint32_t *baseAddr*)
[inline], [static]**

Parameters

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

Returns

The transmit packets 1024 byte to 2047-byte.

**11.2.5.101 static uint16_t ENET_HAL_GetTxOverByte2048Packet (uint32_t *baseAddr*)
[inline], [static]**

Parameters

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

Returns

The transmit packets greater than 2048-bytes.

**11.2.5.102 static uint32_t ENET_HAL_GetTxOctets (uint32_t *baseAddr*) [inline],
[static]**

Parameters

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

Returns

The transmit Octets.

ENET HAL driver

**11.2.5.103 static uint16_t ENET_HAL_GetTxFramesOk(uint32_t *baseAddr*) [inline],
[static]**

Parameters

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

Returns

The Frames transmitted well.

11.2.5.104 static uint16_t ENET_HAL_GetTxFramesOneCollision (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The frames transmitted with single collision.

11.2.5.105 static uint16_t ENET_HAL_GetTxFramesMultiCollision (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The frames transmitted with multiple collision.

11.2.5.106 static uint16_t ENET_HAL_GetTxFrameCarrSenseError (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The frames transmitted with carrier sense error.

11.2.5.107 **static uint16_t ENET_HAL_GetTxFramesDelay (uint32_t *baseAddr*)**
[**inline**], [**static**]

Parameters

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

Returns

The frames transmitted after deferral delay

11.2.5.108 static uint16_t ENET_HAL_GetTxFramesLateCollision (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The frames transmitted with late collision.

11.2.5.109 static uint16_t ENET_HAL_GetTxFramesExcessiveCollision (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The frames transmitted with excessive collisions.

11.2.5.110 static uint16_t ENET_HAL_GetTxFramesMacError (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The Frames transmitted with the Tx FIFO underrun.

**11.2.5.111 static uint16_t ENET_HAL_GetTxFramesPause (uint32_t *baseAddr*)
[inline], [static]**

Parameters

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

Returns

The transmitted flow control Pause Frames.

11.2.5.112 static uint32_t ENET_HAL_GetTxOctetFramesOk (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The octet count for frames transmitted without error.

11.2.5.113 static uint16_t ENET_HAL_GetRxPackets (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packet count.

11.2.5.114 static uint16_t ENET_HAL_GetRxBroadCastPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive broadcast packet count.

11.2.5.115 **static uint16_t ENET_HAL_GetRxMultiCastPacket (uint32_t *baseAddr*)**
[**inline**], [**static**]

Parameters

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

Returns

The receive multicast packet count.

11.2.5.116 static uint16_t ENET_HAL_GetRxCrcAlignErrorPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets with CRC/Align error.

11.2.5.117 static uint16_t ENET_HAL_GetRxUnderSizePacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets less than 64-byte and good CRC.

11.2.5.118 static uint16_t ENET_HAL_GetRxOverSizePacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets greater than MAX_FL and good CRC.

**11.2.5.119 static uint16_t ENET_HAL_GetRxFragPacket (uint32_t *baseAddr*)
[inline], [static]**

Parameters

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

Returns

The receive packets less than 64-byte and bad CRC.

11.2.5.120 static uint16_t ENET_HAL_GetRxJabPacket (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets greater than MAX_FL and bad CRC.

11.2.5.121 static uint16_t ENET_HAL_GetRxByte64Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets with 64-byte.

11.2.5.122 static uint16_t ENET_HAL_GetRxByte65to127Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets with 65-byte to 127-byte.

11.2.5.123 **static uint16_t ENET_HAL_GetRxByte128to255Packet (uint32_t *baseAddr*)**
[**inline**], [**static**]

Parameters

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

Returns

The receive packets with 128-byte to 255-byte.

11.2.5.124 static uint16_t ENET_HAL_GetRxByte256to511Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets with 256-byte to 511-byte.

11.2.5.125 static uint16_t ENET_HAL_GetRxByte512to1023Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets with 512-byte to 1023-byte.

11.2.5.126 static uint16_t ENET_HAL_GetRxByte1024to2047Packet (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive packets with 1024-byte to 2047-byte.

11.2.5.127 **static uint16_t ENET_HAL_GetRxOverByte2048Packet (uint32_t *baseAddr*)**
[**inline**], [**static**]

Parameters

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

Returns

The receive packets greater than 2048.

11.2.5.128 static uint32_t ENET_HAL_GetRxOctets (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The receive octets.

11.2.5.129 static uint16_t ENET_HAL_GetRxFramesDrop (uint32_t *baseAddr*) [inline], [static]

If a frame with invalid or missing SFD character is detected and has been dropped.

Parameters

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

Returns

The receive Frames not counted correctly.

11.2.5.130 static uint16_t ENET_HAL_GetRxFramesOk (uint32_t *baseAddr*) [inline], [static]

ENET HAL driver

Parameters

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

Returns

The Frames received OK.

11.2.5.131 static uint16_t ENET_HAL_GetRxFramesCrcError (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The Frames received with CRC error.

11.2.5.132 static uint16_t ENET_HAL_GetRxFramesAlignError (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The Frames received with Alignment error.

11.2.5.133 static uint16_t ENET_HAL_GetRxFramesMacError (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The FIFO overflow count.

11.2.5.134 **static uint16_t ENET_HAL_GetRxFramesFlowControl (uint32_t *baseAddr*)**
[**inline**], [**static**]

ENET HAL driver

Parameters

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

Returns

The received flow control Pause frames.

11.2.5.135 static uint32_t ENET_HAL_GetRxOtetsFramesOk (uint32_t *baseAddr*) [inline], [static]

Parameters

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

Returns

The octet count for frames received without error.

Chapter 12

External Watchdog Timer (EWM)

12.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the external Watchdog Timer (EWM) block of Kinetis devices.

Modules

- [EWM HAL driver](#)

12.2 EWM HAL driver

12.2.1 Overview

The section describes the programming interface of the EWM HAL driver.

Data Structures

- union `ewm_common_config_t`
ewm common config structure. [More...](#)

Enumerations

- enum `ewm_in_assertion_state_t` {
 `kEWMLogicZeroAssert` = 0,
 `kEWMLogicOneAssert` = 1 }
EWM_in's assertion state.
- enum `ewm_status_t` {
 `kStatus_EWM_Success` = 0x0U,
 `kStatus_EWM_NotInitialized` = 0x1U,
 `kStatus_EWM_NullArgument` = 0x2U }
ewm status return codes.

Functions

- static void `EWM_HAL_SetCommonConfig` (uint32_t baseAddr, `ewm_common_config_t` commonConfig)
Config EWM control register.
- static bool `EWM_HAL_IsEnabled` (uint32_t baseAddr)
Checks whether the EWM is enabled.
- static `ewm_in_assertion_state_t` `EWM_HAL_GetAssertionStateMode` (uint32_t baseAddr)
Checks EWM_in assertion state.
- static bool `EWM_HAL_GetInputCmd` (uint32_t baseAddr)
Checks if EWM_in input is enabled.
- static bool `EWM_HAL.GetIntCmd` (uint32_t baseAddr)
Checks if EWM interrupt is enabled.
- static void `EWM_HAL_SetIntCmd` (uint32_t baseAddr, bool enable)
Enable/Disable EWM interrupt.
- static void `EWM_HAL_SetCmpLowRegValue` (uint32_t baseAddr, uint8_t minServiceCycles)
Set EWM compare low register value.
- static uint8_t `EWM_HAL_GetCmpLowRegValue` (uint32_t baseAddr)
Return EWM compare low register value.
- static void `EWM_HAL_SetCmpHighRegValue` (uint32_t baseAddr, uint8_t maxServiceCycles)
Set EWM compare high register value.
- static uint8_t `EWM_HAL_GetCmpHighRegValue` (uint32_t baseAddr)
Return EWM compare high register value.

- static void [EWM_HAL_Refresh](#) (uint32_t baseAddr)
Service EWM.
- void [EWM_HAL_Init](#) (uint32_t baseAddr)
Restores the EWM module to reset value.

12.2.2 Data Structure Documentation

12.2.2.1 union `ewm_common_config_t`

12.2.3 Enumeration Type Documentation

12.2.3.1 enum `ewm_in_assertion_state_t`

Enumerator

kEWMLogicZeroAssert assert state of EWM_in signal is logic zero
kEWMLogicOneAssert assert state of EWM_in signal is logic one

12.2.3.2 enum `ewm_status_t`

Enumerator

kStatus_EWM_Success Succeed.
kStatus_EWM_NotInitialized EWM is not initialized yet.
kStatus_EWM_NullArgument Argument is NULL.

12.2.4 Function Documentation

12.2.4.1 static void `EWM_HAL_SetCommonConfig` (`uint32_t baseAddr,` `ewm_common_config_t commonConfig`) [inline], [static]

This function configures EWM control register, EWM enable bitfeild, EWM ASSIN bitfeild and EWM INPUT enable bitfeild are WRITE ONCE, one more write will cause bus fault.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

EWM HAL driver

<i>commonConfig</i>	config EWM CTRL register
---------------------	--------------------------

12.2.4.2 static bool EWM_HAL_IsEnabled (uint32_t *baseAddr*) [inline], [static]

This function checks whether the EWM is enabled.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Return values

<i>false</i>	means EWM is disabled
<i>true</i>	means WODG is enabled

12.2.4.3 static ewm_in_assertion_state_t EWM_HAL_GetAssertionStateMode (uint32_t *baseAddr*) [inline], [static]

This function checks EWM_in assertion state.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Return values

<i>kEWMLogicZeroAssert</i>	means assert state of EWM_in signal is logic zero
<i>kEWMLogicOneAssert</i>	means assert state of EWM_in signal is logic one

12.2.4.4 static bool EWM_HAL_GetInputCmd (uint32_t *baseAddr*) [inline], [static]

This function checks whether the EWM_in is enabled.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Return values

<i>true</i>	means EWM_in input is enable
<i>false</i>	means EWM_in input is disable

12.2.4.5 static bool EWM_HAL_GetIntCmd (uint32_t *baseAddr*) [inline], [static]

This function checks whether the EWM interrupt is enabled.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Return values

<i>true</i>	means EWM_in interrupt is enable
<i>false</i>	means EWM_in interrupt is disable

12.2.4.6 static void EWM_HAL_SetIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function sets EWM enable/disable.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
<i>enable</i>	Set EWM interrupt enable/disable

12.2.4.7 static void EWM_HAL_SetCmpLowRegValue (uint32_t *baseAddr*, uint8_t *minServiceCycles*) [inline], [static]

This function sets EWM compare low register value and defines the minimum cycles to service EWM, when counter value is greater than or equal to ewm compare low register value, refresh EWM can be successful, and this register is write once, one more write will cause bus fault.

EWM HAL driver

Parameters

<i>baseAddr</i>	The EWM peripheral base address
<i>minService-Cycles</i>	The EWM compare low register value

12.2.4.8 static uint8_t EWM_HAL_GetCmpLowRegValue (uint32_t *baseAddr*) [inline], [static]

This function return compare low register value.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Return values

<i>compare</i>	low register value
----------------	--------------------

12.2.4.9 static void EWM_HAL_SetCmpHighRegValue (uint32_t *baseAddr*, uint8_t *maxServiceCycles*) [inline], [static]

This function sets EWM compare high register value and defines the maximum cycles to service EWM, when counter value is less than or equal to ewm compare high register value, refresh EWM can be successful, the compare high register value must be greater than compare low register value, and this register is write once, one more write will cause bus fault.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
<i>maxService-Cycles</i>	The EWM compare low register value

12.2.4.10 static uint8_t EWM_HAL_GetCmpHighRegValue (uint32_t *baseAddr*) [inline], [static]

This function return compare high register value, the maximum value is 0xfe, otherwise EWM counter never expires.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Return values

<i>compare</i>	high register value
----------------	---------------------

12.2.4.11 static void EWM_HAL_Refresh (uint32_t *baseAddr*) [inline], [static]

This function reset EWM counter to zero and the period of writing the frist value and the second value should be within 15 bus cycles.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

12.2.4.12 void EWM_HAL_Init (uint32_t *baseAddr*)

This function restores the EWM module to reset value.

Parameters

<i>baseAddr</i>	The EWM peripheral base address
-----------------	---------------------------------

Chapter 13

C90TFS Flash Driver

13.1 Overview

The Kinetis SDK provides C90TFS flash driver of Kinetis devices with C90TFS flash module inside. The C90TFS/FTFx SSD provides general APIs to handle specific operations on C90TFS/FTFx flash module. User can use those APIs directly in his application. In addition, it provides internal functions called by driver itself and they are not intended to call from user's application directly but user still can use those APIs for his purpose.

The C90TFS/FTFx SSD provides the following features:

1. Drivers released in source code format to provide compiler-independent supporting for non-debug-mode embedded applications.
2. Each driver function is independent to each other. So, the end user can choose the function subset to meet their particular needs.
3. Position-independent and ROM-able.
4. Concurrency support via callback.

Important Note

1. The DebugEnable field of [FLASH_SSD_CONFIG](#) structure shall allow user to use this driver in background debug mode, without returning to the calling function, but returning to debug mode instead. To enable this feature, DebugEnable must be set to TRUE and macro C90TFS_ENABLE_DEBUG must be set to 1.
2. If user utilizes callback in his application, this callback function must not be placed in the same flash block in which a program/erase operation is going on to avoid RWW error.
3. If user wants to suspend the sector erase operation, for simple method, just invoke FlashEraseSuspend function within callback of FlashEraseSector. In this case, FlashEraseSuspend must not be placed in the same block in which flash erase sector command is going on.
4. FlashCommandSequence, FlashSuspend and FlashResume should be executed from RAM or different flash blocks which are targeted for writing to avoid RWW error.
5. To guarantee the correct execution of this driver, the flash cache in the flash memory controller module should be disabled before invoking any API. The normal demo included in the release package provides the code part to disable/enable this flash cache as well.

Data Structures

- struct [PFLASH_SSD_CONFIG](#)
Flash SSD Configuration Structure. [More...](#)

Macros

- #define [SET_FLASH_INT_BITS](#)(ftfxRegBase, value)

Overview

- Set the Flash interrupt enable bits in the FCNFG register.
• `#define GET_FLASH_INT_BITS(ftfxRegBase)`
Return the Flash interrupt enable bits in the FCNFG register.

C90TFS Flash configuration

- `#define ENDIANNESS LITTLE_ENDIAN`
Endianness.
- `#define CPU_CORE ARM_CORTEX_M`
cpu core
- `#define FTFx_PSECTOR_SIZE FSL_FEATURE_FLASH_PFLASH_BLOCK_SECTOR_SIZE`
P-Flash sector size.
- `#define FTFx_DSECTOR_SIZE FSL_FEATURE_FLASH_FLEX_NVM_BLOCK_SECTOR_SIZE`
D-Flash sector size.
- `#define DEBLOCK_SIZE (FSL_FEATURE_FLASH_FLEX_NVM_BLOCK_SIZE * FSL_FEATURE_FLASH_NVM_BLOCK_COUNT)`
FlexNVM block size.
- `#define EEESIZE_0000 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0000`
Emulated eeprom size code 0000 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0001 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0001`
Emulated eeprom size code 0001 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0010 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0010`
Emulated eeprom size code 0010 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0011 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0011`
Emulated eeprom size code 0011 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0100 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0100`
Emulated eeprom size code 0100 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0101 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0101`
Emulated eeprom size code 0101 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0110 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0110`
Emulated eeprom size code 0110 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_0111 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_0111`
Emulated eeprom size code 0111 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_1000 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_1000`
Emulated eeprom size code 1000 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_1001 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_1001`
Emulated eeprom size code 1001 mapping to emulated eeprom size in bytes (0xFFFF = reserved)
- `#define EEESIZE_1010 FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESIZE_1010`
Emulated eeprom size code 1010 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

ZE_1010

Emulated eeprom size code 1010 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

- #define **EEESIZE_1011** FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESI-ZE_1011

Emulated eeprom size code 1011 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

- #define **EEESIZE_1100** FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESI-ZE_1100

Emulated eeprom size code 1100 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

- #define **EEESIZE_1101** FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESI-ZE_1101

Emulated eeprom size code 1101 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

- #define **EEESIZE_1110** FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESI-ZE_1110

Emulated eeprom size code 1110 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

- #define **EEESIZE_1111** FSL_FEATURE_FLASH_FLEX_NVM_EEPROM_SIZE_FOR_EEESI-ZE_1111

Emulated eeprom size code 1111 mapping to emulated eeprom size in bytes (0xFFFF = reserved)

- #define **DEPART_0000** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0000

FlexNVM partition code 0000 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0001** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0001

FlexNVM partition code 0001 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0010** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0010

FlexNVM partition code 0010 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0011** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0011

FlexNVM partition code 0011 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0100** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0100

FlexNVM partition code 0100 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0101** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0101

FlexNVM partition code 0101 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0110** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0110

FlexNVM partition code 0110 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_0111** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_0111

FlexNVM partition code 0111 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_1000** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1000

FlexNVM partition code 1000 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_1001** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1001

FlexNVM partition code 1001 mapping to data flash size in bytes (0xFFFFFFFF = reserved)

- #define **DEPART_1010** FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1010

Overview

- **#define DEPART_1011 FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1011**
FlexNVM partition code 1010 mapping to data flash size in bytes (0xFFFFFFFF = reserved)
- **#define DEPART_1100 FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1100**
FlexNVM partition code 1011 mapping to data flash size in bytes (0xFFFFFFFF = reserved)
- **#define DEPART_1101 FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1101**
FlexNVM partition code 1100 mapping to data flash size in bytes (0xFFFFFFFF = reserved)
- **#define DEPART_1110 FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1110**
FlexNVM partition code 1101 mapping to data flash size in bytes (0xFFFFFFFF = reserved)
- **#define DEPART_1111 FSL_FEATURE_FLASH_FLEX_NVM_DFLASH_SIZE_FOR_DEPART_1111**
FlexNVM partition code 1110 mapping to data flash size in bytes (0xFFFFFFFF = reserved)
- **#define DFLASH_IFR_READRESOURCE_ADDRESS 0x8000FCU**
Data flash IFR map.
- **#define PGMCHK_ALIGN_SIZE FSL_FEATURE_FLASH_PFLASH_CHECK_CMD_ADDRESS_ALIGNMENT**
P-Flash Program check command address alignment.
- **#define PGM_SIZE_BYTE FSL_FEATURE_FLASH_PFLASH_BLOCK_WRITE_UNIT_SIZE**
P-Flash write unit size.
- **#define RESUME_WAIT_CNT 0x20U**
Resume wait count used in FlashResume function.

Address convert macros

- **#define BYTE2WORD(x) (x)**
Convert from byte address to word(2 bytes) address.
- **#define WORD2BYTE(x) (x)**
Convert from word(2 bytes) address to byte address.

PFlash swap control codes

- **#define FTFx_SWAP_SET_INDICATOR_ADDR 0x01U**
Initialize Swap System control code.
- **#define FTFx_SWAP_SET_IN_PREPARE 0x02U**
Set Swap in Update State.
- **#define FTFx_SWAP_SET_IN_COMPLETE 0x04U**
Set Swap in Complete State.
- **#define FTFx_SWAP_REPORT_STATUS 0x08U**
Report Swap Status.

PFlash swap states

- **#define FTFx_SWAP_UNINIT 0x00U**
Uninitialized swap mode.
- **#define FTFx_SWAP_READY 0x01U**
Ready swap mode.

- #define **FTFx_SWAP_UPDATE** 0x02U
Update swap mode.
- #define **FTFx_SWAP_UPDATE_ERASED** 0x03U
Update-Erased swap mode.
- #define **FTFx_SWAP_COMPLETE** 0x04U
Complete swap mode.

C90TFS Flash driver APIs

- uint32_t **RelocateFunction** (uint32_t dest, uint32_t size, uint32_t src)
Relocate a function to RAM address.
- uint32_t **FlashInit** (PFLASH_SSD_CONFIG pSSDConfig)
Flash initialization.
- uint32_t **FlashCommandSequence** (PFLASH_SSD_CONFIG pSSDConfig)
Flash command sequence.
- uint32_t **PFlashGetProtection** (PFLASH_SSD_CONFIG pSSDConfig, uint32_t *protectStatus)
P-Flash get protection.
- uint32_t **PFlashSetProtection** (PFLASH_SSD_CONFIG pSSDConfig, uint32_t protectStatus)
P-Flash set protection.
- uint32_t **FlashGetSecurityState** (PFLASH_SSD_CONFIG pSSDConfig, uint8_t *securityState)
Flash get security state.
- uint32_t **FlashSecurityBypass** (PFLASH_SSD_CONFIG pSSDConfig, uint8_t *keyBuffer, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash security bypass.
- uint32_t **FlashEraseAllBlock** (PFLASH_SSD_CONFIG pSSDConfig, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash erase all Blocks.
- uint32_t **FlashVerifyAllBlock** (PFLASH_SSD_CONFIG pSSDConfig, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash verify all Blocks.
- uint32_t **FlashEraseSector** (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash erase sector.
- uint32_t **FlashVerifySection** (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint16_t number, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash verify sector.
- uint32_t **FlashEraseSuspend** (PFLASH_SSD_CONFIG pSSDConfig)
Flash erase suspend.
- uint32_t **FlashEraseResume** (PFLASH_SSD_CONFIG pSSDConfig)
Flash erase resume.
- uint32_t **FlashReadOnce** (PFLASH_SSD_CONFIG pSSDConfig, uint8_t recordIndex, uint8_t *pDataArray, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash read once.
- uint32_t **FlashProgramOnce** (PFLASH_SSD_CONFIG pSSDConfig, uint8_t recordIndex, uint8_t *pDataArray, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash program once.
- uint32_t **FlashReadResource** (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint8_t *pDataArray, uint8_t resourceSelectCode, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
Flash read resource.

Overview

- `uint32_t FlashProgram (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint8_t *pData, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
Flash program.
- `uint32_t FlashProgramCheck (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint8_t *pExpectedData, uint32_t *pFailAddr, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
Flash program check.
- `uint32_t FlashChecksum (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint32_t *pSum)`
Calculate check sum.
- `uint32_t FlashProgramSection (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint16_t number, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
Flash program part.
- `32_t FlashEraseBlock (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
Flash erase block.
- `uint32_t FlashVerifyBlock (PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
Flash verify block.

Return Code Definition for FTFx SSD

- `#define FTFx_OK 0x0000U`
Function executes successfully.
- `#define FTFx_ERR_MGSTAT0 0x0001U`
MGSTAT0 bit is set in the FSTAT register.
- `#define FTFx_ERR_PVIOL 0x0010U`
Protection violation is set in FSTAT register.
- `#define FTFx_ERR_ACCERR 0x0020U`
Access error is set in the FSTAT register.
- `#define FTFx_ERR_CHANGEPROT 0x0100U`
Can not change protection status.
- `#define FTFx_ERR_NOEEE 0x0200U`
FlexRAM is not set for EEPROM use.
- `#define FTFx_ERR_EFLASHONLY 0x0400U`
FlexNVM is set for full EEPROM backup.
- `#define FTFx_ERR_RAMRDY 0x0800U`
Programming acceleration RAM is not available.
- `#define FTFx_ERR_RANGE 0x1000U`
Address is out of the valid range.
- `#define FTFx_ERR_SIZE 0x2000U`
Misaligned size.

Flash security status

- `#define FLASH_NOT_SECURE 0x01U`
Flash currently not in secure state.
- `#define FLASH_SECURE_BACKDOOR_ENABLED 0x02U`
Flash is secured and backdoor key access enabled.
- `#define FLASH_SECURE_BACKDOOR_DISABLED 0x04U`

Flash is secured and backdoor key access disabled.

Null Callback function definition

- #define **NULL_CALLBACK** ((**PCALLBACK**)0xFFFFFFFF)
Null callback.
- #define **NULL_SWAP_CALLBACK** ((**PFLASH_SWAP_CALLBACK**)0xFFFFFFFF)
Null swap callback.

Type definition for flash driver

- typedef void(* **PCALLBACK**)(void)
Call back function pointer data type.
- typedef bool(* **PFLASH_SWAP_CALLBACK**)(uint8_t function)
Swap call back function pointer data type.
- typedef uint32_t(* **pFLASHCOMMANDSEQUENCE**)(PFLASH_SSD_CONFIG pSSDConfig)
FlashCommandSequence function poiter.
- typedef uint32_t(* **pFLASHINIT**)(PFLASH_SSD_CONFIG pSSDConfig)
FlashInit function poiter.
- typedef uint32_t(* **pPFLASHGETPROTECTION**)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t *protectStatus)
PFlashGetProtection function poiter.
- typedef uint32_t(* **pPFLASHSETPROTECTION**)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t protectStatus)
PFlashSetProtection function poiter.
- typedef uint32_t(* **pFLASHGETSECURITYSTATE**)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t *securityState)
FlashGetSecurityState function poiter.
- typedef uint32_t(* **pFLASHSECURITYBYPASS**)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t *keyBuffer, **pFLASHCOMMANDSEQUENCE** pFlashCommandSequence)
FlashSecurityByPass function poiter.
- typedef uint32_t(* **pFLASHERASEALLBLOCK**)(PFLASH_SSD_CONFIG pSSDConfig, **pFLASHCOMMANDSEQUENCE** pFlashCommandSequence)
FlashEraseAllBlock function poiter.
- typedef uint32_t(* **pFLASHERASEBLOCK**)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, **pFLASHCOMMANDSEQUENCE** pFlashCommandSequence)
FlashEraseBlock function poiter.
- typedef uint32_t(* **pFLASHERASESECTOR**)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, **pFLASHCOMMANDSEQUENCE** pFlashCommandSequence)
FlashEraseSector function poiter.
- typedef uint32_t(* **pFLASHERASESUSPEND**)(PFLASH_SSD_CONFIG pSSDConfig)
FlashEraseSuspend function poiter.
- typedef uint32_t(* **pFLASHERASERESUME**)(PFLASH_SSD_CONFIG pSSDConfig)
FlashEraseResume function poiter.
- typedef uint32_t(* **pFLASHPROGRAMSECTION**)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint16_t number, **pFLASHCOMMANDSEQUENCE** pFlashCommandSequence)
FlashProgramSection function poiter.
- typedef uint32_t(* **pFLASHCHECKSUM**)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint32_t *pSum)
FlashChecksum function poiter.

Overview

- `typedef uint32_t(* pFLASHVERIFYALLBLOCK)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashVerifyAllBlock function poiter.
- `typedef uint32_t(* pFLASHVERIFYBLOCK)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
Flash verify block.
- `typedef uint32_t(* pFLASHVERIFYSECTION)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint16_t number, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashVerifySection function poiter.
- `typedef uint32_t(* pFLASHREADONCE)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t *pDataArray, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashReadOnce function poiter.
- `typedef uint32_t(* pFLASHPROGRAMONCE)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t *pDataArray, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashProgramOnce function poiter.
- `typedef uint32_t(* pFLASHPROGRAMCHECK)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint8_t *pExpectedData, uint32_t *pFailAddr, uint8_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashProgramCheck function poiter.
- `typedef uint32_t(* pFLASHREADRESOURCE)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint8_t *pDataArray, uint8_t resourceSelectCode, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashReadResource function poiter.
- `typedef uint32_t(* pFLASHPROGRAM)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint8_t *pData, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
FlashProgram function poiter.
- `typedef uint32_t(* pPFLASHSWAPCTRL)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t addr, uint8_t swapcmd, uint8_t *pCurrentSwapMode, uint8_t *pCurrentSwapBlockStatus, uint8_t *pNextSwapBlockStatus, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
PFlashSwapCtrl function poiter.
- `typedef uint32_t(* pFLASHSWAP)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t flashAddress, PFLASH_SWAP_CALLBACK pSwapCallback, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`
PFlashSwap function poiter.
- `typedef uint32_t(* pDFLASHGETPROTECTION)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t *protectStatus)`
DFlashGetProtection function poiter.
- `typedef uint32_t(* pDFLASHSETPROTECTION)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t protectStatus)`
DFlashSetProtection function poiter.
- `typedef uint32_t(* pEERAMGETPROTECTION)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t *protectStatus)`
EERAMGetProtection function poiter.
- `typedef uint32_t(* pEERAMSETPROTECTION)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t protectStatus)`
EERAMSetProtection function poiter.
- `typedef uint32_t(* pDEFLASHPARTITION)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t E-`

`EEDataSizeCode, uint8_t DEPartitionCode, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`

DEFlashPartition function poiter.

- `typedef uint32_t(* pSETEEEENABLE)(PFLASH_SSD_CONFIG pSSDConfig, uint8_t EEEEnable, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`

SetEEEEnable function poiter.

- `typedef uint32_t(* pEEEWRITE)(PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, uint32_t size, uint8_t *pData)`

EEEWrite function poiter.

13.2 Data Structure Documentation

13.2.1 struct FLASH_SSD_CONFIG

The structure includes the static parameters for C90TFS/FTFx which are device-dependent. The user should correctly initialize the fields including fftxRegBase, PFlashBlockBase, PFlashBlockSize, DFlashBlockBase, EERAMBlockBase, DebugEnable and CallBack before passing the structure to SSD functions. The rest of parameters such as DFlashBlockSize, and EEEBlockSize will be initialized in [Flash-Init\(\)](#) automatically. The pointer to CallBack has to be initialized either for null callback or a valid call back function.

Data Fields

- `uint32_t fftxRegBase`
The register base address of C90TFS/FTFx.
- `uint32_t PFlashBase`
The base address of P-Flash memory.
- `uint32_t PFlashSize`
The size in byte of P-Flash memory.
- `uint32_t DFlashBase`
For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory); For non-FlexNVM device, this field is unused.
- `uint32_t DFlashSize`
For FlexNVM device, this is the size in byte of area which is used as D-Flash from FlexNVM memory; For non-FlexNVM device, this field is unused.
- `uint32_t EERAMBase`
The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device)
- `uint32_t EEESize`
For FlexNVM device, this is the size in byte of EEPROM area which was partitioned from FlexRAM; For non-FlexNVM device, this field is unused.
- `bool DebugEnable`
Background debug mode enable.
- `PCALLBACK CallBack`
Call back function to service the time critical events.

Macro Definition Documentation

13.3 Macro Definition Documentation

13.3.1 #define BYTE2WORD(x)(x)

Two address types are only different in DSC devices. In Kinstis devices, they are the same

13.3.2 #define WORD2BYTE(x)(x)

Two address types are only different in DSC devices. In Kinstis devices, they are the same

13.3.3 #define SET_FLASH_INT_BITS(*ftfxRegBase*, *value*)

Value:

```
REG_WRITE( (ftfxRegBase) + FTFx_SSD_FCNFG_OFFSET, \
           ((value) & (FTFx_SSD_FCNFG_CCIE | FTFx_SSD_FCNFG_RDCOLLIE)) )
```

Parameters

<i>ftfxRegBase</i> :	Specify register base address of flash module
<i>value</i> :	The bit map value (0: disabled, 1 enabled) . The numbering is marked from 0 to 7 where bit 0 is the least significant bit. Bit 7 is corresponding to command complete interrupt. Bit 6 is corresponding to read collision error interrupt.

13.3.4 #define GET_FLASH_INT_BITS(*ftfxRegBase*)

Value:

```
REG_READ( (ftfxRegBase) + FTFx_SSD_FCNFG_OFFSET) & \
          (FTFx_SSD_FCNFG_CCIE | FTFx_SSD_FCNFG_RDCOLLIE)
```

Parameters

<i>ftfxRegBase</i> :	Specify register base address of flash module
----------------------	---

13.3.5 #define FTFx_ERR_MGSTAT0 0x0001U

Possible causes:

MGSTAT0 bit in FSTAT register is set. Refer to corresponding command description of each API on reference manual to get detail reasons

Solution:

Hardware error

13.3.6 #define FTFx_ERR_PVIOL 0x0010U

Possible causes:

FPIVOL bit in FSTAT register is set. Refer to corresponding command description of each API on reference manual to get detail reasons

Solution:

The flash location targeted to program/erase operation must be unprotected. Swap indicator must not be programmed/erased except in Update or Update-Erase state.

13.3.7 #define FTFx_ERR_ACCERR 0x0020U

Possible causes:

ACCERR bit in FSTAT register is set. Refer to corresponding command description of each API on reference manual to get detail reasons.

Solution:

Provide valid input parameters for each API according to specific flash module.

13.3.8 #define FTFx_ERR_CHANGEPROT 0x0100U

Possible causes:

Violate protection transition

Solution:

In NVM normal mode, protection size cannot be decreased. So, only increasing protection size is permitted if the device is operating in this mode.

13.3.9 #define FTFx_ERR_NOEEE 0x0200U

Possible causes:

User accesses to EEPROM operation but there is no EEPROM backup enabled.

Solution:

Need to enable EEPROM by partitioning FlexNVM to have EEPROM backup and/or enable it by SetEE-Enable API.

Function Documentation

13.3.10 #define FTFx_ERR_EFLASHONLY 0x0400U

Possible causes:

User accesses to D-Flash operation but there is no D-Flash on FlexNVM.

Solution:

Need to partition FlexNVM to have D-Flash.

13.3.11 #define FTFx_ERR_RAMRDY 0x0800U

Possible causes:

User invokes flash program part command but FlexRam is being set for EEPROM emulation. Solution:

Need to set FlexRam as traditional Ram by SetEEEnable API.

13.3.12 #define FTFx_ERR_RANGE 0x1000U

Possible causes:

The size or destination provided by user makes start address or end address out of valid range.

Solution:

Make sure the destination and (destination + size) within valid address range.

13.3.13 #define FTFx_ERR_SIZE 0x2000U

Possible causes:

The size provided by user is misaligned.

Solution:

Size must be an aligned value according to specific constrain of each API.

13.4 Function Documentation

13.4.1 uint32_t RelocateFunction (uint32_t dest, uint32_t size, uint32_t src)

This function provides users a facility to relocate a function from one location to another location in RAM.

Parameters

<i>dest,:</i>	Destination address where you want to place the function .
<i>size,:</i>	Size of the function
<i>src,:</i>	Address of the function will be relocated

Returns

Relocated address of the function .

13.4.2 **uint32_t FlashInit (PFLASH_SSD_CONFIG *pSSDConfig*)**

This API will initialize flash module by clearing status error bit and reporting the memory configuration via SSD configuration structure.

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
---------------------	--

Returns

Successful completion (FTFx_OK)

13.4.3 **uint32_t FlashCommandSequence (PFLASH_SSD_CONFIG *pSSDConfig*)**

This API is used to perform command write sequence on the flash. It is internal function, called by driver APIs only

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
---------------------	--

Returns

Successful completion (FTFx_OK)

Failed in flash command execution (FTFx_ERR_ACCERR, FTFx_ERR_PVIOL, FTFx_ERR_MGSTAT0)

Function Documentation

13.4.4 uint32_t PFlashGetProtection (**PFLASH_SSD_CONFIG pSSDConfig,** **uint32_t * protectStatus)**

This API retrieves current P-Flash protection status. Considering the time consumption for getting protection is very low and even can be ignored, it is not necessary to utilize the Callback function to support the time-critical events.

Parameters

<i>pSSDConfig</i> ,:	The SSD configuration structure pointer.
<i>protectStatus</i> ,:	To return the current value of the P-Flash Protection. Each bit is corresponding to protection of 1/32 of the total P-Flash. The least significant bit is corresponding to the lowest address area of P-Flash. The most significant bit is corresponding to the highest address area of P- Flash and so on. There are two possible cases as below: <ul style="list-style-type: none">• 0: this area is protected.• 1: this area is unprotected.

Returns

Successful completion (FTFx_OK)

13.4.5 uint32_t PFlashSetProtection (**PFLASH_SSD_CONFIG pSSDConfig,** **uint32_t protectStatus)**

This API sets the P-Flash protection to the intended protection status. Setting P-Flash protection status is subject to a protection transition restriction. If there is any setting violation, it will return an error code and the current protection status won't be changed.

Parameters

<i>pSSDConfig</i> ,:	The SSD configuration structure pointer.
<i>protectStatus</i> ,:	The expected protect status user wants to set to P-Flash protection register. Each bit is corresponding to protection of 1/32 of the total P-Flash. The least significant bit is corresponding to the lowest address area of P-Flash. The most significant bit is corresponding to the highest address area of P- Flash and so on. There are two possible cases as below: <ul style="list-style-type: none">• 0: this area is protected.• 1: this area is unprotected.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_CHANGEPROT)

13.4.6 uint32_t FlashGetSecurityState (PFLASH_SSD_CONFIG pSSDConfig, uint8_t * securityState)

This API retrieves the current Flash security status, including the security enabling state and the back door key enabling state.

Parameters

<i>pSSDConfig</i> :	The SSD configuration structure pointer.
<i>securityState</i> :	To return the current security status code. FLASH_NOT_SECURE (0x01): Flash currently not in secure state; FLASH_SECURE_BACKDOOR_ENABLED (0x02): Flash is secured and back door key access enabled; FLASH_SECURE_BACKDOOR_DISABLED (0x04): Flash is secured and back door key access disabled.

Returns

Successful completion (FTFx_OK)

13.4.7 uint32_t FlashSecurityBypass (PFLASH_SSD_CONFIG pSSDConfig, uint8_t * keyBuffer, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)

This API will unsecure the device by comparing the user's provided back door key with the ones in the Flash Configuration Field. If they are matched each other, then security will be released. Otherwise, an error code will be returned.

Parameters

<i>pSSDConfig</i> :	The SSD configuration structure pointer.
<i>keyBuffer</i> :	Point to the user buffer containing the back door key.
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_ACCERR)

Function Documentation

13.4.8 `uint32_t FlashEraseAllBlock (PFLASH_SSD_CONFIG pSSDConfig, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)`

This API will erase all Flash memory, initialize the FlexRAM, verify all memory contents, and then release MCU security

Parameters

<i>pSSDConfig</i> :	The SSD configuration structure pointer.
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_PVIOL, FTFx_ERR_MGSTAT0, FTFx_ERR_ACCERR)

13.4.9 **uint32_t FlashVerifyAllBlock (PFLASH_SSD_CONFIG *pSSDConfig*, uint8_t *marginLevel*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)**

This function will check to see if the P-Flash and/or D-Flash, EEPROM backup area, and D-Flash IFR have been erased to the specified read margin level, if applicable, and will release security if the readout passes

Parameters

<i>pSSDConfig</i> :	The SSD configuration structure pointer.
<i>marginLevel</i> :	Read Margin Choice as follows: marginLevel = 0x0: use the Normal read level marginLevel = 0x1: use the User read marginLevel = 0x2: use the Factory read
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_MGSTAT0, FTFx_ERR_ACCERR)

13.4.10 **uint32_t FlashEraseSector (PFLASH_SSD_CONFIG *pSSDConfig*, uint32_t *dest*, uint32_t *size*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)**

This API will erase one or more sectors in P-Flash or D-Flash memory. This API always returns FTFx_OK if size provided by user is zero regardless of the input validation.

Function Documentation

Parameters

<i>pSSDConfig</i> ,	The SSD configuration structure pointer.
<i>dest</i> ,	Address in the first sector to be erased.
<i>size</i> ,	Size to be erased in bytes. It is used to determine number of sectors to be erased.
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)

Error value (FTFx_ERR_MGSTAT0, FTFx_ERR_ACCERR, FTFx_ERR_PVIOL, FTFx_ERR_SI-ZE)

13.4.11 **uint32_t FlashVerifySection (PFLASH_SSD_CONFIG *pSSD-Config*, uint32_t *dest*, uint16_t *number*, uint8_t *marginLevel*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)**

This API will check to see if a part of P-Flash or D-Flash memory is erased to the specified read margin level

Parameters

<i>pSSDConfig</i> ,	The SSD configuration structure pointer.
<i>dest</i> ,	Start address for the intended verify operation.
<i>number</i> ,	Number of alignment unit to be verified. Refer to corresponding reference manual to get correct information of alignment constrain.
<i>marginLevel</i> ,	Read Margin Choice as follows: marginLevel = 0x0: use Normal read level margin-Level = 0x1: use the User read marginLevel = 0x2: use the Factory read
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)

Error value (FTFx_ERR_MGSTAT0, FTFx_ERR_ACCERR)

13.4.12 `uint32_t FlashEraseSuspend (PFLASH_SSD_CONFIG pSSDConfig)`

This API is used to suspend a current operation of flash erase sector command. This function must be located in RAM memory or different flash blocks which are targeted for writing to avoid RWW error

Function Documentation

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
---------------------	--

Returns

Successful completion (FTFx_OK)

13.4.13 uint32_t FlashEraseResume (PFLASH_SSD_CONFIG pSSDConfig)

This API is used to resume a previous suspended operation of flash erase sector command. This function must be located in RAM memory or different flash blocks which are targeted for writing to avoid RWW error.

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
---------------------	--

Returns

Successful completion (FTFx_OK)

13.4.14 uint32_t FlashReadOnce (PFLASH_SSD_CONFIG pSSDConfig, uint8_t recordIndex, uint8_t * pDataArray, pFLASHCOMMANDSEQUENCE pFlashCommandSequence)

This API is used to read out a reserved 64 byte field located in the P-Flash IFR via given number of record. Refer to corresponding reference manual to get correct value of this number.

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
<i>recordIndex,:</i>	The record index will be read. It can be from 0x0 to 0x7 or from 0x0 to 0xF according to specific derivative.
<i>pDataArray,:</i>	Pointer to the array to return the data read by the read once command.

<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.
--------------------------------	---

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_ACCERR)

13.4.15 **uint32_t FlashProgramOnce (PFLASH_SSD_CONFIG *pSSDConfig*, uint8_t *recordIndex*, uint8_t * *pdataArray*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)**

This API is used to program to a reserved 64 byte field located in the P-Flash IFR via given number of record. Refer to corresponding reference manual to get correct value of this number.

Parameters

<i>pSSDConfig</i> :	The SSD configuration structure pointer.
<i>recordIndex</i> :	The record index will be read. It can be from 0x0 to 0x7 or from 0x0 to 0xF according to specific derivative.
<i>pdataArray</i> :	Pointer to the array from which data will be taken for program once command.
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_ACCERR,FTFx_ERR_MGSTAT0)

13.4.16 **uint32_t FlashReadResource (PFLASH_SSD_CONFIG *pSSDConfig*, uint32_t *dest*, uint8_t * *pdataArray*, uint8_t *resourceSelectCode*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)**

This API is used to read data from special purpose memory in flash memory module including P-Flash IFR, swap IFR, D-Flash IFR space and version ID.

Function Documentation

Parameters

<i>pSSDConfig</i> ,:	The SSD configuration structure pointer.
<i>dest</i> ,:	Start address for the intended read operation.
<i>pdataArray</i> ,:	Pointer to the data returned by the read resource command.
<i>resourceSelectCode</i> ,:	Read resource select code: 0 : Flash IFR 1: Version ID
<i>pFlashCommandSequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_ACCERR)

13.4.17 uint32_t FlashProgram (PFLASH_SSD_CONFIG *pSSDConfig*, uint32_t *dest*, uint32_t *size*, uint8_t * *pData*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)

This API is used to program 4 consecutive bytes (for program long word command) and 8 consecutive bytes (for program phrase command) on P-flash or D-Flash block. This API always returns FTFx_OK if size provided by user is zero regardless of the input validation

Parameters

<i>pSSDConfig</i> ,:	The SSD configuration structure pointer.
<i>dest</i> ,:	Start address for the intended program operation.
<i>size</i> ,:	Size in byte to be programmed
<i>pData</i> ,:	Pointer of source address from which data has to be taken for program operation.
<i>pFlashCommandSequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_ACCERR, FTFx_ERR_PVIOL, FTFx_ERR_SIZE, FTFx_ERR_MGST-AT0)

**13.4.18 uint32_t FlashProgramCheck (PFLASH_SSD_CONFIG *pSSDConfig*,
 uint32_t *dest*, uint32_t *size*, uint8_t * *pExpectedData*, uint32_t *
pFailAddr, uint8_t *marginLevel*, pFLASHCOMMANDSEQUENCE
pFlashCommandSequence)**

This API tests a previously programmed P-Flash or D-Flash long word to see if it reads correctly at the specified margin level. This API always returns FTFx_OK if size provided by user is zero regardless of the input validation

Parameters

<i>pSSDConfig</i> :	The SSD configuration structure pointer.
<i>dest</i> :	Start address for the intended program check operation.
<i>size</i> :	Size in byte to check accuracy of program operation
<i>pExpected-Data</i> :	The pointer to the expected data.
<i>pFailAddr</i> :	Returned the first aligned failing address.
<i>marginLevel</i> :	Read margin choice as follows: marginLevel = 0x1: read at User margin 1/0 level. marginLevel = 0x2: read at Factory margin 1/0 level.
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
 Error value (FTFx_ERR_ACCERR, FTFx_ERR_MGSTAT0)

**13.4.19 uint32_t FlashCheckSum (PFLASH_SSD_CONFIG *pSSDConfig*, uint32_t
dest, uint32_t *size*, uint32_t * *pSum*)**

This API will perform 32 bit sum of each byte data over specified flash memory range without carry, which provides rapid method for checking data integrity. The callback time period of this API is determined via FLASH_CALLBACK_CS macro in SSD_FTFx_Common.h which is used as a counter value for the CallBack() function calling in this API. This value can be changed as per the user requirement. User can change this value to obtain the maximum permissible callback time period. This API always returns FTFx_OK if size provided by user is zero regardless of the input validation.

Function Documentation

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
<i>dest,:</i>	Start address of the Flash range to be summed
<i>size,:</i>	Size in byte of the flash range to be summed
<i>pSum,:</i>	To return the sum value

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_RANGE)

**13.4.20 uint32_t FlashProgramSection (PFLASH_SSD_CONFIG *pSSDConfig*,
 uint32_t *dest*, uint16_t *number*, pFLASHCOMMANDSEQUENCE
 pFlashCommandSequence)**

This API will program the data found in the Section Program Buffer to previously erased locations in the Flash memory. Data is preloaded into the Section Program Buffer by writing to the acceleration Ram and FlexRam while it is set to function as a RAM. The Section Program Buffer is limited to the value of FlexRam divides by a ratio. Refer to the associate reference manual to get correct value of this ratio. For derivatives including swap feature, the swap indicator address is encountered during FlashProgramSection, it is bypassed without setting FPVIOL but the content are not be programmed. In addition, the content of source data used to program to swap indicator will be re-initialized to 0xFF after completion of this command.

Parameters

<i>pSSDConfig,:</i>	The SSD configuration structure pointer.
<i>dest,:</i>	Start address for the intended program operation.
<i>number,:</i>	Number of alignment unit to be programmed. Refer to associate reference manual to get correct value of this alignment constrain.
<i>pFlash- Comman- dSeque- nce</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)
Error value (FTFx_ERR_ACCERR, FTFx_ERR_PVIOL, FTFx_ERR_MGSTAT0, FTFx_ERR_R-AMRDY)

13.4.21 uint32_t FlashEraseBlock (PFLASH_SSD_CONFIG *pSSDConfig*, uint32_t *dest*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)

This API will erase all addresses in an individual P-Flash or D-Flash block. For the derivatives including multiply logical P-Flash or D-Flash blocks, this API just erases a single block in a single call.

Function Documentation

Parameters

<i>pSSDConfig</i> ,	The SSD configuration structure pointer.
<i>dest</i> ,	Start address for the intended erase operation.
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)

Error value (FTFx_ERR_ACCERR, FTFx_ERR_PVIOL, FTFx_ERR_MGSTAT0)

13.4.22 uint32_t FlashVerifyBlock (PFLASH_SSD_CONFIG *pSSDConfig*, uint32_t *dest*, uint8_t *marginLevel*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence*)

This API will check to see if an entire P-Flash or D-Flash block has been erased to the specified margin level For the derivatives including multiply logical P-Flash or D-Flash blocks, this API just erases a single block in a single call.

Parameters

<i>pSSDConfig</i> ,	The SSD configuration structure pointer.
<i>dest</i> ,	Start address for the intended verify operation.
<i>marginLevel</i> ,	Read Margin Choice as follows: marginLevel = 0x0: use Normal read level margin- Level = 0x1: use the User read marginLevel = 0x2: use the Factory read
<i>pFlash-Command-Sequence</i>	: Pointer to the flash command sequence function.

Returns

Successful completion (FTFx_OK)

Error value (FTFx_ERR_ACCERR, FTFx_ERR_MGSTAT0)

Chapter 14

Controller Area Network (FlexCAN)

14.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the FlexCAN Controller Area Network (FlexCAN) block of Kinetis devices.

Modules

- [FlexCAN Driver](#)
- [FlexCAN HAL driver](#)

14.2 FlexCAN HAL driver

14.2.1 Overview

This section describes the programming interface of the FlexCAN HAL driver.

Data Structures

- struct `flexcan_id_table_t`
FlexCAN RX FIFO ID filter table structure. [More...](#)
- struct `flexcan_berr_counter_t`
FlexCAN bus error counters. [More...](#)
- struct `flexcan_mb_code_status_t`
FlexCAN MB code and status for transmit and receive. [More...](#)
- struct `flexcan_mb_t`
FlexCAN message buffer structure. [More...](#)
- struct `flexcan_time_segment_t`
FlexCAN timing related structures. [More...](#)

Enumerations

- enum `_flexcan_constants` { `kFlexCanMessageSize` = 8 }
FlexCAN constants.
- enum `_flexcan_err_status` {
 `kFlexCan_RxWrn` = 0x0080,
 `kFlexCan_TxWrn` = 0x0100,
 `kFlexCan_StfErr` = 0x0200,
 `kFlexCan_FrmErr` = 0x0400,
 `kFlexCan_CrcErr` = 0x0800,
 `kFlexCan_AckErr` = 0x1000,
 `kFlexCan_Bit0Err` = 0x2000,
 `kFlexCan_Bit1Err` = 0x4000 }
- The Status enum is used to report current status of the FlexCAN interface.*
- enum `flexcan_status_t`
FlexCAN status return codes.
- enum `flexcan_operation_modes_t` {
 `kFlexCanNormalMode`,
 `kFlexCanListenOnlyMode`,
 `kFlexCanLoopBackMode`,
 `kFlexCanFreezeMode`,
 `kFlexCanDisableMode` }
- FlexCAN operation modes.*
- enum `flexcan_mb_code_rx_t` {

```
kFlexCanRX_Inactive = 0x0,
kFlexCanRX_Full = 0x2,
kFlexCanRX_Empty = 0x4,
kFlexCanRX_Overrun = 0x6,
kFlexCanRX_Busy = 0x8,
kFlexCanRX_Ranswer = 0xA,
kFlexCanRX_NotUsed = 0xF }
```

FlexCAN message buffer CODE for Rx buffers.

- enum `flexcan_mb_code_tx_t` {
 kFlexCanTX_Inactive = 0x08,
 kFlexCanTX_Abort = 0x09,
 kFlexCanTX_Data = 0x0C,
 kFlexCanTX_Remote = 0x1C,
 kFlexCanTX_Tanswer = 0x0E,
 kFlexCanTX_NotUsed = 0xF }

FlexCAN message buffer CODE FOR Tx buffers.

- enum `flexcan_mb_transmission_type_t` {
 kFlexCanMBStatusType_TX,
 kFlexCanMBStatusType_TXRemote,
 kFlexCanMBStatusType_RX,
 kFlexCanMBStatusType_RXRemote,
 kFlexCanMBStatusType_RXTXRemote }

FlexCAN message buffer transmission types.

- enum `flexcan_rx_fifo_id_element_format_t` {
 kFlexCanRxFifoIdElementFormat_A,
 kFlexCanRxFifoIdElementFormat_B,
 kFlexCanRxFifoIdElementFormat_C,
 kFlexCanRxFifoIdElementFormat_D }
- enum `flexcan_rx_fifo_id_filter_num_t` {
 kFlexCanRxFifoIDFilters_8 = 0x0,
 kFlexCanRxFifoIDFilters_16 = 0x1,
 kFlexCanRxFifoIDFilters_24 = 0x2,
 kFlexCanRxFifoIDFilters_32 = 0x3,
 kFlexCanRxFifoIDFilters_40 = 0x4,
 kFlexCanRxFifoIDFilters_48 = 0x5,
 kFlexCanRxFifoIDFilters_56 = 0x6,
 kFlexCanRxFifoIDFilters_64 = 0x7,
 kFlexCanRxFifoIDFilters_72 = 0x8,
 kFlexCanRxFifoIDFilters_80 = 0x9,
 kFlexCanRxFifoIDFilters_88 = 0xA,
 kFlexCanRxFifoIDFilters_96 = 0xB,
 kFlexCanRxFifoIDFilters_104 = 0xC,
 kFlexCanRxFifoIDFilters_112 = 0xD,
 kFlexCanRxFifoIDFilters_120 = 0xE,
 kFlexCanRxFifoIDFilters_128 = 0xF }

FlexCAN Rx FIFO filters number.

FlexCAN HAL driver

- enum `flexcan_rx_mask_type_t` {
 `kFlexCanRxMask_Global`,
 `kFlexCanRxMask_Individual` }
 FlexCAN RX mask type.
- enum `flexcan_mb_id_type_t` {
 `kFlexCanMbId_Std`,
 `kFlexCanMbId_Ext` }
 FlexCAN MB ID type.
- enum `flexcan_clk_source_t` {
 `kFlexCanClkSource_Osc`,
 `kFlexCanClkSource_Ipbus` }
 FlexCAN clock source.
- enum `flexcan_int_type_t` {
 `kFlexCanInt_Buf`,
 `kFlexCanInt_Err`,
 `kFlexCanInt_Boff`,
 `kFlexCanInt_Wakeup`,
 `kFlexCanInt_Txwarning`,
 `kFlexCanInt_Rxwarning` }
 FlexCAN error interrupt types.

Configuration

- `flexcan_status_t FLEXCAN_HAL_Enable` (`uint32_t canBaseAddr`)
 Enables FlexCAN controller.
- `flexcan_status_t FLEXCAN_HAL_Disable` (`uint32_t canBaseAddr`)
 Disables FlexCAN controller.
- static bool `FLEXCAN_HAL_IsEnabled` (`uint32_t canBaseAddr`)
 Checks whether the FlexCAN is enabled or disabled.
- `flexcan_status_t FLEXCAN_HAL_SelectClock` (`uint32_t canBaseAddr, flexcan_clk_source_t clk`)
 Selects the clock source for FlexCAN.
- static bool `FLEXCAN_HAL_GetClock` (`uint32_t canBaseAddr`)
 Reads the clock source for FlexCAN Protocol Engine (PE).
- `flexcan_status_t FLEXCAN_HAL_Init` (`uint32_t canBaseAddr`)
 Initializes the FlexCAN controller.
- void `FLEXCAN_HAL_SetTimeSegments` (`uint32_t canBaseAddr, flexcan_time_segment_t *time_seg`)
 Sets the FlexCAN time segments for setting up bit rate.
- void `FLEXCAN_HAL_GetTimeSegments` (`uint32_t canBaseAddr, flexcan_time_segment_t *time_seg`)
 Gets the FlexCAN time segments to calculate the bit rate.
- void `FLEXCAN_HAL_ExitFreezeMode` (`uint32_t canBaseAddr`)
 Unfreezes the FlexCAN module.
- void `FLEXCAN_HAL_EnterFreezeMode` (`uint32_t canBaseAddr`)
 Freezes the FlexCAN module.
- `flexcan_status_t FLEXCAN_HAL_EnableOperationMode` (`uint32_t canBaseAddr, flexcan_operation_modes_t mode`)
 Enables operation mode.

- **flexcan_status_t FLEXCAN_HAL_DisableOperationMode** (uint32_t canBaseAddr, **flexcan_operation_modes_t** mode)
- Disables operation mode.*

Data transfer

- **flexcan_status_t FLEXCAN_HAL_SetMbTx** (uint32_t canBaseAddr, uint32_t mb_idx, **flexcan_mb_code_status_t** *cs, uint32_t msg_id, uint8_t *mb_data)
- Sets the FlexCAN message buffer fields for transmitting.*
- **flexcan_status_t FLEXCAN_HAL_SetMbRx** (uint32_t canBaseAddr, uint32_t mb_idx, **flexcan_mb_code_status_t** *cs, uint32_t msg_id)
- Sets the FlexCAN message buffer fields for receiving.*
- **flexcan_status_t FLEXCAN_HAL_GetMb** (uint32_t canBaseAddr, uint32_t mb_idx, **flexcan_mb_t** *mb)
- Gets the FlexCAN message buffer fields.*
- **flexcan_status_t FLEXCAN_HAL_LockRxMb** (uint32_t canBaseAddr, uint32_t mb_idx)
- Locks the FlexCAN Rx message buffer.*
- static void **FLEXCAN_HAL_UnlockRxMb** (uint32_t canBaseAddr)
- Unlocks the FlexCAN Rx message buffer.*
- void **FLEXCAN_HAL_EnableRxFifo** (uint32_t canBaseAddr, uint32_t numOfFilters, uint32_t maxNumMb)
- Enables the Rx FIFO.*
- void **FLEXCAN_HAL_DisableRxFifo** (uint32_t canBaseAddr)
- Disables the Rx FIFO.*
- void **FLEXCAN_HAL_SetRxFifoFiltersNumber** (uint32_t canBaseAddr, uint32_t number)
- Sets the number of the Rx FIFO filters.*
- void **FLEXCAN_HAL_SetMaxMbNumber** (uint32_t canBaseAddr, uint32_t maxNumMb)
- Sets the maximum number of Message Buffers.*
- **flexcan_status_t FLEXCAN_HAL_SetIdFilterTableElements** (uint32_t canBaseAddr, **flexcan_rx_fifo_id_element_format_t** id_format, **flexcan_id_table_t** *id_filter_table)
- Sets the Rx FIFO ID filter table elements.*
- **flexcan_status_t FLEXCAN_HAL_SetRxFifo** (uint32_t canBaseAddr, **flexcan_rx_fifo_id_element_format_t** id_format, **flexcan_id_table_t** *id_filter_table)
- Sets the FlexCAN Rx FIFO fields.*
- **flexcan_status_t FLEXCAN_HAL_ReadFifo** (uint32_t canBaseAddr, **flexcan_mb_t** *rx_fifo)
- Gets the FlexCAN Rx FIFO data.*

Interrupts

- **flexcan_status_t FLEXCAN_HAL_EnableMbInt** (uint32_t canBaseAddr, uint32_t mb_idx)
- Enables the FlexCAN Message Buffer interrupt.*
- **flexcan_status_t FLEXCAN_HAL_DisableMbInt** (uint32_t canBaseAddr, uint32_t mb_idx)
- Disables the FlexCAN Message Buffer interrupt.*
- void **FLEXCAN_HAL_EnableErrInt** (uint32_t canBaseAddr)
- Enables error interrupt of the FlexCAN module.*
- void **FLEXCAN_HAL_DisableErrInt** (uint32_t canBaseAddr)
- Disables error interrupt of the FlexCAN module.*

FlexCAN HAL driver

- void **FLEXCAN_HAL_EnableBusOffInt** (uint32_t canBaseAddr)
Enables Bus off interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_DisableBusOffInt** (uint32_t canBaseAddr)
Disables Bus off interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_EnableWakeupInt** (uint32_t canBaseAddr)
Enables Wakeup interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_DisableWakeupInt** (uint32_t canBaseAddr)
Disables Wakeup interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_EnableTxWarningInt** (uint32_t canBaseAddr)
Enables TX warning interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_DisableTxWarningInt** (uint32_t canBaseAddr)
Disables TX warning interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_EnableRxWarningInt** (uint32_t canBaseAddr)
Enables RX warning interrupt of the FlexCAN module.
- void **FLEXCAN_HAL_DisableRxWarningInt** (uint32_t canBaseAddr)
Disables RX warning interrupt of the FlexCAN module.

Status

- static uint32_t **FLEXCAN_HAL_GetFreezeAck** (uint32_t canBaseAddr)
Gets the value of FlexCAN freeze ACK.
- uint8_t **FLEXCAN_HAL_GetMbIntFlag** (uint32_t canBaseAddr, uint32_t mb_idx)
Gets the individual FlexCAN MB interrupt flag.
- static uint32_t **FLEXCAN_HAL_GetAllMbIntFlags** (uint32_t canBaseAddr)
Gets all FlexCAN MB interrupt flags.
- static void **FLEXCAN_HAL_ClearMbIntFlag** (uint32_t canBaseAddr, uint32_t reg_val)
Clears the interrupt flag of the message buffers.
- void **FLEXCAN_HAL_GetErrCounter** (uint32_t canBaseAddr, flexcan_berr_counter_t *err_cnt)
Gets the transmit error counter and receives the error counter.
- static uint32_t **FLEXCAN_HAL_GetErrStatus** (uint32_t canBaseAddr)
Gets error and status.
- void **FLEXCAN_HAL_ClearErrIntStatus** (uint32_t canBaseAddr)
Clears all other interrupts in ERRSTAT register (Error, Busoff, Wakeup).

Mask

- void **FLEXCAN_HAL_SetMaskType** (uint32_t canBaseAddr, flexcan_rx_mask_type_t type)
Sets the Rx masking type.
- void **FLEXCAN_HAL_SetRx_fifoGlobalStdMask** (uint32_t canBaseAddr, uint32_t std_mask)
Sets the FlexCAN RX FIFO global standard mask.
- void **FLEXCAN_HAL_SetRx_fifoGlobalExtMask** (uint32_t canBaseAddr, uint32_t ext_mask)
Sets the FlexCAN Rx FIFO global extended mask.
- flexcan_status_t **FLEXCAN_HAL_SetRxIndividualStdMask** (uint32_t canBaseAddr, uint32_t mb_idx, uint32_t std_mask)
Sets the FlexCAN Rx individual standard mask for ID filtering in the Rx MBs and the Rx FIFO.
- flexcan_status_t **FLEXCAN_HAL_SetRxIndividualExtMask** (uint32_t canBaseAddr, uint32_t mb_idx, uint32_t ext_mask)
Sets the FlexCAN Rx individual extended mask for ID filtering in the Rx MBs and the Rx FIFO.

- void **FLEXCAN_HAL_SetRxMbGlobalStdMask** (uint32_t canBaseAddr, uint32_t std_mask)
Sets the FlexCAN Rx MB global standard mask.
- void **FLEXCAN_HAL_SetRxMbBuf14StdMask** (uint32_t canBaseAddr, uint32_t std_mask)
Sets the FlexCAN RX MB BUF14 standard mask.
- void **FLEXCAN_HAL_SetRxMbBuf15StdMask** (uint32_t canBaseAddr, uint32_t std_mask)
Sets the FlexCAN Rx MB BUF15 standard mask.
- void **FLEXCAN_HAL_SetRxMbGlobalExtMask** (uint32_t canBaseAddr, uint32_t ext_mask)
Sets the FlexCAN RX MB global extended mask.
- void **FLEXCAN_HAL_SetRxMbBuf14ExtMask** (uint32_t canBaseAddr, uint32_t ext_mask)
Sets the FlexCAN RX MB BUF14 extended mask.
- void **FLEXCAN_HAL_SetRxMbBuf15ExtMask** (uint32_t canBaseAddr, uint32_t ext_mask)
Sets the FlexCAN RX MB BUF15 extended mask.
- static uint32_t **FLEXCAN_HAL_GetIdAcceptanceFilterRxFifo** (uint32_t canBaseAddr)
Gets the FlexCAN ID acceptance filter hit indicator on Rx FIFO.

14.2.2 Data Structure Documentation

14.2.2.1 struct flexcan_id_table_t

Data Fields

- bool **is_remote_mb**
Remote frame.
- bool **is_extended_mb**
Extended frame.
- uint32_t * **id_filter**
Rx FIFO ID filter elements.

14.2.2.2 struct flexcan_berr_counter_t

Data Fields

- uint16_t **txerr**
Transmit error counter.
- uint16_t **rxerr**
Receive error counter.

14.2.2.3 struct flexcan_mb_code_status_t

Data Fields

- uint32_t **code**
MB code for TX or RX buffers.
- **flexcan_mb_id_type_t msg_id_type**
Type of message ID (standard or extended)
- uint32_t **data_length**
Length of Data in Bytes.

FlexCAN HAL driver

14.2.2.3.0.30 Field Documentation

14.2.2.3.0.30.1 uint32_t flexcan_mb_code_status_t::code

Defined by flexcan_mb_code_rx_t and flexcan_mb_code_tx_t

14.2.2.4 struct flexcan_mb_t

Data Fields

- uint32_t **cs**
Code and Status.
- uint32_t **msg_id**
Message Buffer ID.
- uint8_t **data** [kFlexCanMessageSize]
Bytes of the FlexCAN message.

14.2.2.5 struct flexcan_time_segment_t

Data Fields

- uint32_t **propseg**
Propagation segment.
- uint32_t **pseg1**
Phase segment 1.
- uint32_t **pseg2**
Phase segment 2.
- uint32_t **pre_divider**
Clock pre divider.
- uint32_t **rjw**
Resync jump width.

14.2.3 Enumeration Type Documentation

14.2.3.1 enum _flexcan_constants

Enumerator

kFlexCanMessageSize FlexCAN message buffer data size in bytes.

14.2.3.2 enum _flexcan_err_status

Enumerator

kFlexCan_RxWrn Reached warning level for RX errors.

kFlexCan_TxWrn Reached warning level for TX errors.

kFlexCan_StfErr Stuffing Error.
kFlexCan_FrmErr Form Error.
kFlexCan_CrcErr Cyclic Redundancy Check Error.
kFlexCan_AckErr Received no ACK on transmission.
kFlexCan_Bit0Err Unable to send dominant bit.
kFlexCan_Bit1Err Unable to send recessive bit.

14.2.3.3 enum flexcan_operation_modes_t

Enumerator

kFlexCanNormalMode Normal mode or user mode.
kFlexCanListenOnlyMode Listen-only mode.
kFlexCanLoopBackMode Loop-back mode.
kFlexCanFreezeMode Freeze mode.
kFlexCanDisableMode Module disable mode.

14.2.3.4 enum flexcan_mb_code_rx_t

Enumerator

kFlexCanRX_Inactive MB is not active.
kFlexCanRX_Full MB is full.
kFlexCanRX_Empty MB is active and empty.
kFlexCanRX_Overrun MB is overwritten into a full buffer.
kFlexCanRX_Busy FlexCAN is updating the contents of the MB.
kFlexCanRX_Ranswer The CPU must not access the MB. A frame was configured to recognize a
 Remote Request Frame
kFlexCanRX_NotUsed and transmit a Response Frame in return. Not used

14.2.3.5 enum flexcan_mb_code_tx_t

Enumerator

kFlexCanTX_Inactive MB is not active.
kFlexCanTX_Abort MB is aborted.
kFlexCanTX_Data MB is a TX Data Frame(MB RTR must be 0).
kFlexCanTX_Remote MB is a TX Remote Request Frame (MB RTR must be 1).
kFlexCanTX_Tanswer MB is a TX Response Request Frame from.
kFlexCanTX_NotUsed an incoming Remote Request Frame. Not used

14.2.3.6 enum flexcan_mb_transmission_type_t

Enumerator

kFlexCanMBStatusType_TX Transmit MB.

kFlexCanMBStatusType_TXRemote Transmit remote request MB.

kFlexCanMBStatusType_RX Receive MB.

kFlexCanMBStatusType_RXRemote Receive remote request MB.

kFlexCanMBStatusType_RXTXRemote FlexCAN remote frame receives remote request and.

14.2.3.7 enum flexcan_rx_fifo_id_element_format_t

Enumerator

kFlexCanRxFifoIdElementFormat_A One full ID (standard and extended) per ID Filter Table.

kFlexCanRxFifoIdElementFormat_B element. Two full standard IDs or two partial 14-bit (standard and

kFlexCanRxFifoIdElementFormat_C extended) IDs per ID Filter Table element. Four partial 8-bit Standard IDs per ID Filter Table

kFlexCanRxFifoIdElementFormat_D element. All frames rejected.

14.2.3.8 enum flexcan_rx_fifo_id_filter_num_t

Enumerator

kFlexCanRxFifoIDFilters_8 8 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_16 16 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_24 24 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_32 32 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_40 40 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_48 48 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_56 56 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_64 64 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_72 72 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_80 80 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_88 88 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_96 96 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_104 104 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_112 112 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_120 120 Rx FIFO Filters.

kFlexCanRxFifoIDFilters_128 128 Rx FIFO Filters.

14.2.3.9 enum flexcan_rx_mask_type_t

Enumerator

kFlexCanRxMask_Global Rx global mask.*kFlexCanRxMask_Individual* Rx individual mask.**14.2.3.10 enum flexcan_mb_id_type_t**

Enumerator

kFlexCanMbId_Std Standard ID.*kFlexCanMbId_Ext* Extended ID.**14.2.3.11 enum flexcan_clk_source_t**

Enumerator

kFlexCanClkSource_Osc Oscillator clock.*kFlexCanClkSource_Ipbus* Peripheral clock.**14.2.3.12 enum flexcan_int_type_t**

Enumerator

kFlexCanInt_Buf OR'd message buffers interrupt.*kFlexCanInt_Err* Error interrupt.*kFlexCanInt_Boff* Bus off interrupt.*kFlexCanInt_Wakeup* Wakeup interrupt.*kFlexCanInt_Txwarning* TX warning interrupt.*kFlexCanInt_Rxwarning* RX warning interrupt.**14.2.4 Function Documentation****14.2.4.1 flexcan_status_t FLEXCAN_HAL_Enable (uint32_t canBaseAddr)**

Parameters

FlexCAN HAL driver

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

0 if successful; non-zero failed

14.2.4.2 **flexcan_status_t FLEXCAN_HAL_Disable (uint32_t canBaseAddr)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

0 if successful; non-zero failed

14.2.4.3 **static bool FLEXCAN_HAL_IsEnabled (uint32_t canBaseAddr) [inline], [static]**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

State of FlexCAN enable(0)/disable(1)

14.2.4.4 **flexcan_status_t FLEXCAN_HAL_SelectClock (uint32_t canBaseAddr, flexcan_clk_source_t clk)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>clk</i>	The FlexCAN clock source

Returns

0 if successful; non-zero failed

14.2.4.5 **static bool FLEXCAN_HAL_GetClock (uint32_t canBaseAddr) [inline], [static]**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

0: if clock source is oscillator clock, 1: if clock source is peripheral clock

14.2.4.6 **flexcan_status_t FLEXCAN_HAL_Init (uint32_t canBaseAddr)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

0 if successful; non-zero failed

14.2.4.7 **void FLEXCAN_HAL_SetTimeSegments (uint32_t canBaseAddr, flexcan_time_segment_t * time_seg)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>time_seg</i>	FlexCAN time segments, which need to be set for the bit rate.

Returns

0 if successful; non-zero failed

14.2.4.8 **void FLEXCAN_HAL_GetTimeSegments (uint32_t canBaseAddr, flexcan_time_segment_t * time_seg)**

Parameters

FlexCAN HAL driver

<i>canBaseAddr</i>	The FlexCAN base address
<i>time_seg</i>	FlexCAN time segments read for bit rate

Returns

0 if successful; non-zero failed.

14.2.4.9 void FLEXCAN_HAL_ExitFreezeMode (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

0 if successful; non-zero failed.

14.2.4.10 void FLEXCAN_HAL_EnterFreezeMode (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.11 flexcan_status_t FLEXCAN_HAL_EnableOperationMode (uint32_t *canBaseAddr*, flexcan_operation_modes_t *mode*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mode</i>	An operation mode to be enabled

Returns

0 if successful; non-zero failed.

14.2.4.12 flexcan_status_t FLEXCAN_HAL_DisableOperationMode (uint32_t *canBaseAddr*, flexcan_operation_modes_t *mode*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mode</i>	An operation mode to be disabled

Returns

0 if successful; non-zero failed.

14.2.4.13 **flexcan_status_t FLEXCAN_HAL_SetMbTx (uint32_t canBaseAddr, uint32_t mb_idx, flexcan_mb_code_status_t * cs, uint32_t msg_id, uint8_t * mb_data)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer
<i>cs</i>	CODE/status values (TX)
<i>msg_id</i>	ID of the message to transmit
<i>mb_data</i>	Bytes of the FlexCAN message

Returns

0 if successful; non-zero failed

14.2.4.14 **flexcan_status_t FLEXCAN_HAL_SetMbRx (uint32_t canBaseAddr, uint32_t mb_idx, flexcan_mb_code_status_t * cs, uint32_t msg_id)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer
<i>cs</i>	CODE/status values (RX)
<i>msg_id</i>	ID of the message to receive

Returns

0 if successful; non-zero failed

14.2.4.15 **flexcan_status_t FLEXCAN_HAL_GetMb (uint32_t canBaseAddr, uint32_t mb_idx, flexcan_mb_t * mb)**

FlexCAN HAL driver

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer
<i>mb</i>	The fields of the message buffer

Returns

0 if successful; non-zero failed

14.2.4.16 **flexcan_status_t FLEXCAN_HAL_LockRxMb (uint32_t canBaseAddr, uint32_t mb_idx)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer

Returns

0 if successful; non-zero failed

14.2.4.17 **static void FLEXCAN_HAL_UnlockRxMb (uint32_t canBaseAddr) [inline], [static]**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

0 if successful; non-zero failed

14.2.4.18 **void FLEXCAN_HAL_EnableRxFifo (uint32_t canBaseAddr, uint32_t numFilters, uint32_t maxNumMb)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>numOfFilters</i>	The number of Rx FIFO filters
<i>maxNumMb</i>	Maximum number of message buffers

14.2.4.19 void FLEXCAN_HAL_DisableRxFifo (*uint32_t canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.20 void FLEXCAN_HAL_SetRx_fifoFiltersNumber (*uint32_t canBaseAddr, uint32_t number*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>number</i>	The number of Rx FIFO filters

14.2.4.21 void FLEXCAN_HAL_SetMaxMbNumber (*uint32_t canBaseAddr, uint32_t maxNumMb*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>maxNumMb</i>	Maximum number of message buffers

FlexCAN HAL driver

14.2.4.22 flexcan_status_t FLEXCAN_HAL_SetIdFilterTableElements (uint32_t canBaseAddr, flexcan_rx_fifo_id_element_format_t id_format, flexcan_id_table_t * id_filter_table)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>id_format</i>	The format of the Rx FIFO ID Filter Table Elements
<i>id_filter_table</i>	The ID filter table elements which contain if RTR bit, IDE bit and RX message ID need to be set.

Returns

0 if successful; non-zero failed.

14.2.4.23 flexcan_status_t FLEXCAN_HAL_SetRxFifo (uint32_t canBaseAddr, flexcan_rx_fifo_id_element_format_t id_format, flexcan_id_table_t * id_filter_table)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>id_format</i>	The format of the Rx FIFO ID Filter Table Elements
<i>id_filter_table</i>	The ID filter table elements which contain RTR bit, IDE bit, and RX message ID.

Returns

0 if successful; non-zero failed.

14.2.4.24 flexcan_status_t FLEXCAN_HAL_ReadFifo (uint32_t canBaseAddr, flexcan_mb_t * rx_fifo)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>rx_fifo</i>	The FlexCAN receive FIFO data

Returns

0 if successful; non-zero failed.

14.2.4.25 **flexcan_status_t FLEXCAN_HAL_EnableMbInt (uint32_t canBaseAddr, uint32_t mb_idx)**

FlexCAN HAL driver

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer

Returns

0 if successful; non-zero failed

14.2.4.26 **flexcan_status_t FLEXCAN_HAL_DisableMbInt (uint32_t canBaseAddr, uint32_t mb_idx)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer

Returns

0 if successful; non-zero failed

14.2.4.27 **void FLEXCAN_HAL_EnableErrInt (uint32_t canBaseAddr)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.28 **void FLEXCAN_HAL_DisableErrInt (uint32_t canBaseAddr)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.29 **void FLEXCAN_HAL_EnableBusOffInt (uint32_t canBaseAddr)**

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.30 void FLEXCAN_HAL_DisableBusOffInt (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.31 void FLEXCAN_HAL_EnableWakeupInt (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.32 void FLEXCAN_HAL_DisableWakeupInt (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.33 void FLEXCAN_HAL_EnableTxWarningInt (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.34 void FLEXCAN_HAL_DisableTxWarningInt (uint32_t *canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.35 void FLEXCAN_HAL_EnableRxWarningInt (uint32_t *canBaseAddr*)

FlexCAN HAL driver

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.36 void FLEXCAN_HAL_DisableRxWarningInt (*uint32_t canBaseAddr*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.37 static uint32_t FLEXCAN_HAL_GetFreezeAck (*uint32_t canBaseAddr*) [inline], [static]

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

freeze ACK state (1-freeze mode, 0-not in freeze mode).

14.2.4.38 uint8_t FLEXCAN_HAL_GetMbIntFlag (*uint32_t canBaseAddr, uint32_t mb_idx*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer

Returns

the individual MB interrupt flag (0 and 1 are the flag value)

14.2.4.39 static uint32_t FLEXCAN_HAL_GetAllMbIntFlags (*uint32_t canBaseAddr*) [inline], [static]

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

all MB interrupt flags

14.2.4.40 static void FLEXCAN_HAL_ClearMbIntFlag (uint32_t *canBaseAddr*, uint32_t *reg_val*) [inline], [static]

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>reg_val</i>	The value to be written to the interrupt flag1 register.

14.2.4.41 void FLEXCAN_HAL_GetErrCounter (uint32_t *canBaseAddr*, flexcan_berr_counter_t * *err_cnt*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>err_cnt</i>	Transmit error counter and receive error counter

14.2.4.42 static uint32_t FLEXCAN_HAL_GetErrStatus (uint32_t *canBaseAddr*) [inline], [static]

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

The current error and status

14.2.4.43 void FLEXCAN_HAL_ClearErrIntStatus (uint32_t *canBaseAddr*)

FlexCAN HAL driver

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

14.2.4.44 void FLEXCAN_HAL_SetMaskType (uint32_t *canBaseAddr*, flexcan_rx_mask_type_t *type*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>type</i>	The FlexCAN Rx mask type

14.2.4.45 void FLEXCAN_HAL_SetRxFifoGlobalStdMask (uint32_t *canBaseAddr*, uint32_t *std_mask*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>std_mask</i>	Standard mask

14.2.4.46 void FLEXCAN_HAL_SetRxFifoGlobalExtMask (uint32_t *canBaseAddr*, uint32_t *ext_mask*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>ext_mask</i>	Extended mask

14.2.4.47 flexcan_status_t FLEXCAN_HAL_SetRxIndividualStdMask (uint32_t *canBaseAddr*, uint32_t *mb_idx*, uint32_t *std_mask*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer
<i>std_mask</i>	Individual standard mask

Returns

0 if successful; non-zero failed

14.2.4.48 flexcan_status_t FLEXCAN_HAL_SetRxIndividualExtMask (uint32_t canBaseAddr, uint32_t mb_idx, uint32_t ext_mask)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>mb_idx</i>	Index of the message buffer
<i>ext_mask</i>	Individual extended mask

Returns

0 if successful; non-zero failed

14.2.4.49 void FLEXCAN_HAL_SetRxMbGlobalStdMask (uint32_t canBaseAddr, uint32_t std_mask)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>std_mask</i>	Standard mask

14.2.4.50 void FLEXCAN_HAL_SetRxMbBuf14StdMask (uint32_t canBaseAddr, uint32_t std_mask)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>std_mask</i>	Standard mask

14.2.4.51 void FLEXCAN_HAL_SetRxMbBuf15StdMask (uint32_t canBaseAddr, uint32_t std_mask)

FlexCAN HAL driver

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>std_mask</i>	Standard mask

Returns

0 if successful; non-zero failed

14.2.4.52 void FLEXCAN_HAL_SetRxMbGlobalExtMask (uint32_t *canBaseAddr*, uint32_t *ext_mask*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>ext_mask</i>	Extended mask

14.2.4.53 void FLEXCAN_HAL_SetRxMbBuf14ExtMask (uint32_t *canBaseAddr*, uint32_t *ext_mask*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>ext_mask</i>	Extended mask

14.2.4.54 void FLEXCAN_HAL_SetRxMbBuf15ExtMask (uint32_t *canBaseAddr*, uint32_t *ext_mask*)

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
<i>ext_mask</i>	Extended mask

14.2.4.55 static uint32_t FLEXCAN_HAL_GetIdAcceptanceFilterRxFifo (uint32_t *canBaseAddr*) [inline], [static]

Parameters

<i>canBaseAddr</i>	The FlexCAN base address
--------------------	--------------------------

Returns

RX FIFO information

FlexCAN Driver

14.3 FlexCAN Driver

14.3.1 Overview

This section describes the programming interface of the FlexCAN Peripheral driver.

14.3.2 FlexCAN Overview

The FlexCAN (flexible controller area network) module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. The FlexCAN module supports both standard and extended message frames. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module. The CAN Protocol Engine (PE) sub-module manages the serial communication on the CAN bus by requesting RAM access for receiving and transmitting message frames, validating received messages, and performing error handling.

14.3.3 FlexCAN Initialization

To initialize the FlexCAN driver, call the [FLEXCAN_DRV_Init\(\)](#) function and pass the instance number of the FlexCAN you want to use. For example, to use FlexCAN0, pass a value of 0 to the `flexcan_init` function. In addition, you should also pass a user configuration structure [flexcan_user_config_t](#), as shown here:

```
// FlexCAN configuration structure for user
typedef struct FLEXCANUserConfig {
    uint32_t max_num_mb;
    flexcan_rx_fifo_id_filter_num_t num_id_filters;
    bool is_rx_fifo_needed;
} flexcan_user_config_t;
```

Typically, the user configures the [flexcan_user_config_t](#) instantiation as 16 message buffers needed, 8 RX FIFO ID filters and set to true when RX FIFO is needed. This is a code example to set up a FlexCAN configuration instantiation:

```
flexcan_config_t flexcan1_data;
flexcan1_data.max_num_mb = 16;
flexcan1_data.num_id_filters = kFlexCanRx_fifoIDFilters_8;
flexcan1_data.is_rx_fifo_needed = true;
```

14.3.4 FlexCAN Module timing

FlexCAN bit rate is derived from the serial clock, which is generated by dividing the PE clock by the programmed PRESDIV value. Each serial clock period is also called a time quantum. The FlexCAN bit-rate is defined as the Sclock divided by the number of time quanta, where time quanta are further broken down segments within the bit time (time to transmit and sample a bit). The following list shows the CAN

bit-rates that are supported in the FlexCAN driver. 1 Mbytes/s 750 Kbytes/s 500 Kbytes/s 250 Kbytes/s 125 Kbytes/s

The FlexCAN module supports several different ways to set up the bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2, and RJW.

To calculate the CAN bit timing parameters, use the method outlined in the Application Note AN1798, section 4.1. A maximum time for PROP_SEG is used, the remaining TQ is split equally between PSEG1 and PSEG2, provided PSEG2 >=2. RJW is set to the minimum of 4 or to the PSEG1.

14.3.5 FlexCAN Transfers

To transmit a FlexCAN frame, the CPU must prepare a message buffer for transmission by calling the [FLEXCAN_DRV_Send\(\)](#) function. To receive the FlexCAN frames into a message buffer, the CPU must also prepare it for reception by first setting up the receive mask. Functions for this are [FLEXCAN_DRV_SetMaskType\(\)](#), [FLEXCAN_DRV_SetRx_fifoGlobalMask\(\)](#), [FLEXCAN_DRV_SetRxMbGlobalMask\(\)](#) and [FLEXCAN_DRV_SetRxIndividualMask\(\)](#). The user can then configure the RX buffers by calling either [FLEXCAN_DRV_ConfigRxMb\(\)](#) or [FLEXCAN_DRV_ConfigRxFifo\(\)](#) depending on the mode of reception. Finally the user calls [FLEXCAN_DRV_RxMessageBuffer\(\)](#) or [FLEXCAN_DRV_RxFifo](#) functions depending on the mode of reception. The FlexCAN uses only the interrupt-driven process to transfer data.

Data Structures

- struct [flexcan_state_t](#)
Internal driver state information. [More...](#)
- struct [flexcan_data_info_t](#)
FlexCAN data info from user. [More...](#)
- struct [flexcan_user_config_t](#)
FlexCAN configuration. [More...](#)

Functions

- void [FLEXCAN_DRV_IRQHandler](#) (uint8_t instance)
Interrupt handler for a FlexCAN instance.

Variables

- const uint32_t [g_flexcanBaseAddr](#) []
Table of base addresses for FlexCAN instances.
- const IRQn_Type [g_flexcanRxWarningIrqId](#) []
Table to save RX Warning IRQ numbers for FlexCAN instances.
- const IRQn_Type [g_flexcanTxWarningIrqId](#) []

FlexCAN Driver

Table to save TX Warning IRQ numbers for FlexCAN instances.

- const IRQn_Type **g_flexcanWakeUpIrqId** []

Table to save wakeup IRQ numbers for FlexCAN instances.

- const IRQn_Type **g_flexcanErrorIrqId** []

Table to save error IRQ numbers for FlexCAN instances.

- const IRQn_Type **g_flexcanBusOffIrqId** []

Table to save Bus off IRQ numbers for FlexCAN instances.

- const IRQn_Type **g_flexcanOredMessageBufferIrqId** []

Table to save message buffer IRQ numbers for FlexCAN instances.

Bit rate

- **flexcan_status_t FLEXCAN_DRV_SetBitrate** (uint8_t instance, **flexcan_time_segment_t** *bitrate)
Sets the FlexCAN bit rate.
- **flexcan_status_t FLEXCAN_DRV_GetBitrate** (uint8_t instance, **flexcan_time_segment_t** *bitrate)
Gets the FlexCAN bit rate.

Global mask

- void **FLEXCAN_DRV_SetMaskType** (uint8_t instance, **flexcan_rx_mask_type_t** type)
Sets the RX masking type.
- **flexcan_status_t FLEXCAN_DRV_SetRx_fifoGlobalMask** (uint8_t instance, **flexcan_mb_id_type_t** id_type, uint32_t mask)
Sets the FlexCAN RX FIFO global standard or extended mask.
- **flexcan_status_t FLEXCAN_DRV_SetRxMbGlobalMask** (uint8_t instance, **flexcan_mb_id_type_t** id_type, uint32_t mask)
Sets the FlexCAN RX MB global standard or extended mask.
- **flexcan_status_t FLEXCAN_DRV_SetRxIndividualMask** (uint8_t instance, **flexcan_mb_id_type_t** id_type, uint32_t mb_idx, uint32_t mask)
Sets the FlexCAN RX individual standard or extended mask.

Initialization and Shutdown

- **flexcan_status_t FLEXCAN_DRV_Init** (uint8_t instance, const **flexcan_user_config_t** *data, bool enable_err_interrupts, **flexcan_state_t** *state)
Initializes the FlexCAN peripheral.
- **uint32_t FLEXCAN_DRV_Deinit** (uint8_t instance)
Shuts down a FlexCAN instance.

Send configuration

- **flexcan_status_t FLEXCAN_DRV_ConfigTxMb** (uint8_t instance, uint32_t mb_idx, **flexcan_data_info_t** *tx_info, uint32_t msg_id)
FlexCAN transmit message buffer field configuration.

- **flexcan_status_t FLEXCAN_DRV_Send** (uint8_t instance, uint32_t mb_idx, **flexcan_data_info_t** *tx_info, uint32_t msg_id, uint8_t *mb_data, uint32_t timeout_ms)
Sends FlexCAN messages.

Receive configuration

- **flexcan_status_t FLEXCAN_DRV_ConfigRxMb** (uint8_t instance, uint32_t mb_idx, **flexcan_data_info_t** *rx_info, uint32_t msg_id)
FlexCAN receive message buffer field configuration.
- **flexcan_status_t FLEXCAN_DRV_ConfigRxFifo** (uint8_t instance, **flexcan_rx_fifo_id_element_format_t** id_format, **flexcan_id_table_t** *id_filter_table)
FlexCAN RX FIFO field configuration.
- **flexcan_status_t FLEXCAN_DRV_RxMessageBuffer** (uint8_t instance, uint32_t mb_idx, **flexcan_mb_t** *data, uint32_t timeout_ms)
FlexCAN is waiting to receive data from the Message buffer.
- **flexcan_status_t FLEXCAN_DRV_RxFifo** (uint8_t instance, **flexcan_mb_t** *data, uint32_t timeout_ms)
FlexCAN is waiting to receive data from the Message FIFO.

14.3.6 Data Structure Documentation

14.3.6.1 struct flexcan_state_t

Note

The contents of this structure are internal to the driver and should not be modified by users. Also, contents of the structure are subject to change in future releases.

14.3.6.2 struct flexcan_data_info_t

Data Fields

- **flexcan_mb_id_type_t msg_id_type**
Type of message ID (standard or extended)
- **uint32_t data_length**
Length of Data in Bytes.

14.3.6.3 struct flexcan_user_config_t

Data Fields

- **uint32_t max_num_mb**
The maximum number of Message Buffers.
- **flexcan_rx_fifo_id_filter_num_t num_id_filters**

FlexCAN Driver

- *The number of Rx FIFO ID filters needed.*
• bool **is_rx_fifo_needed**
1 if needed; 0 if not

14.3.7 Function Documentation

14.3.7.1 **flexcan_status_t FLEXCAN_DRV_SetBitrate (uint8_t instance, flexcan_time_segment_t * bitrate)**

Parameters

<i>instance</i>	A FlexCAN instance number
<i>bitrate</i>	A pointer to the FlexCAN bit rate settings.

Returns

0 if successful; non-zero failed

14.3.7.2 **flexcan_status_t FLEXCAN_DRV_GetBitrate (uint8_t instance, flexcan_time_segment_t * bitrate)**

Parameters

<i>instance</i>	A FlexCAN instance number
<i>bitrate</i>	A pointer to a variable for returning the FlexCAN bit rate settings

Returns

0 if successful; non-zero failed

14.3.7.3 **void FLEXCAN_DRV_SetMaskType (uint8_t instance, flexcan_rx_mask_type_t type)**

Parameters

<i>instance</i>	A FlexCAN instance number
<i>type</i>	The FlexCAN RX mask type

14.3.7.4 **flexcan_status_t FLEXCAN_DRV_SetRxFifoGlobalMask (uint8_t *instance*, flexcan_mb_id_type_t *id_type*, uint32_t *mask*)**

Parameters

<i>instance</i>	A FlexCAN instance number
<i>id_type</i>	Standard ID or extended ID
<i>mask</i>	Mask value

Returns

0 if successful; non-zero failed

14.3.7.5 **flexcan_status_t FLEXCAN_DRV_SetRxMbGlobalMask (uint8_t *instance*, flexcan_mb_id_type_t *id_type*, uint32_t *mask*)**

Parameters

<i>instance</i>	A FlexCAN instance number
<i>id_type</i>	Standard ID or extended ID
<i>mask</i>	Mask value

Returns

0 if successful; non-zero failed

14.3.7.6 **flexcan_status_t FLEXCAN_DRV_SetRxIndividualMask (uint8_t *instance*, flexcan_mb_id_type_t *id_type*, uint32_t *mb_idx*, uint32_t *mask*)**

Parameters

FlexCAN Driver

<i>instance</i>	A FlexCAN instance number
<i>id_type</i>	A standard ID or an extended ID
<i>mb_idx</i>	Index of the message buffer
<i>mask</i>	Mask value

Returns

0 if successful; non-zero failed.

14.3.7.7 flexcan_status_t FLEXCAN_DRV_Init (uint8_t *instance*, const flexcan_user_config_t * *data*, bool *enable_err_interrupts*, flexcan_state_t * *state*)

This function initializes

Parameters

<i>instance</i>	A FlexCAN instance number
<i>data</i>	The FlexCAN platform data
<i>enable_err_interrupts</i>	1 if enabled, 0 if not
<i>state</i>	Pointer to the FlexCAN driver state structure.

Returns

0 if successful; non-zero failed

14.3.7.8 uint32_t FLEXCAN_DRV_Deinit (uint8_t *instance*)

Parameters

<i>instance</i>	A FlexCAN instance number
-----------------	---------------------------

Returns

0 if successful; non-zero failed

14.3.7.9 flexcan_status_t FLEXCAN_DRV_ConfigTxMb (uint8_t *instance*, uint32_t *mb_idx*, flexcan_data_info_t * *tx_info*, uint32_t *msg_id*)

Parameters

<i>instance</i>	A FlexCAN instance number
<i>mb_idx</i>	Index of the message buffer
<i>tx_info</i>	Data info
<i>msg_id</i>	ID of the message to transmit

Returns

0 if successful; non-zero failed

14.3.7.10 flexcan_status_t FLEXCAN_DRV_Send (uint8_t *instance*, uint32_t *mb_idx*, flexcan_data_info_t * *tx_info*, uint32_t *msg_id*, uint8_t * *mb_data*, uint32_t *timeout_ms*)

Parameters

<i>instance</i>	A FlexCAN instance number
<i>mb_idx</i>	Index of the message buffer
<i>tx_info</i>	Data info
<i>msg_id</i>	ID of the message to transmit
<i>mb_data</i>	Bytes of the FlexCAN message
<i>timeout_ms</i>	A timeout for the transfer in microseconds.

Returns

0 if successful; non-zero failed

14.3.7.11 flexcan_status_t FLEXCAN_DRV_ConfigRxMb (uint8_t *instance*, uint32_t *mb_idx*, flexcan_data_info_t * *rx_info*, uint32_t *msg_id*)

Parameters

FlexCAN Driver

<i>instance</i>	A FlexCAN instance number
<i>mb_idx</i>	Index of the message buffer
<i>rx_info</i>	Data info
<i>msg_id</i>	ID of the message to transmit

Returns

0 if successful; non-zero failed

14.3.7.12 flexcan_status_t FLEXCAN_DRV_ConfigRxFifo (uint8_t *instance*, flexcan_rx_fifo_id_element_format_t *id_format*, flexcan_id_table_t * *id_filter_table*)

Parameters

<i>instance</i>	A FlexCAN instance number
<i>id_format</i>	The format of the Rx FIFO ID Filter Table Elements
<i>id_filter_table</i>	The ID filter table elements which contain RTR bit, IDE bit, and RX message ID

Returns

0 if successful; non-zero failed.

14.3.7.13 flexcan_status_t FLEXCAN_DRV_RxMessageBuffer (uint8_t *instance*, uint32_t *mb_idx*, flexcan_mb_t * *data*, uint32_t *timeout_ms*)

Parameters

<i>instance</i>	A FlexCAN instance number
<i>mb_idx</i>	Index of the message buffer
<i>data</i>	The FlexCAN receive message buffer data.
<i>timeout_ms</i>	A timeout for the transfer in microseconds.

Returns

0 if successful; non-zero failed

14.3.7.14 flexcan_status_t FLEXCAN_DRV_RxFifo (uint8_t *instance*, flexcan_mb_t * *data*, uint32_t *timeout_ms*)

Parameters

<i>instance</i>	A FlexCAN instance number
<i>data</i>	The FlexCAN receive message buffer data.
<i>timeout_ms</i>	A timeout for the transfer in microseconds.

Returns

0 if successful; non-zero failed

14.3.7.15 void FLEXCAN_DRV_IRQHandler (uint8_t *instance*)

Parameters

<i>instance</i>	The FlexCAN instance number.
-----------------	------------------------------

14.3.8 Variable Documentation

14.3.8.1 const uint32_t g_flexcanBaseAddr[]

14.3.8.2 const IRQn_Type g_flexcanRxWarningIrqlId[]

14.3.8.3 const IRQn_Type g_flexcanTxWarningIrqlId[]

14.3.8.4 const IRQn_Type g_flexcanWakeUpIrqlId[]

14.3.8.5 const IRQn_Type g_flexcanErrorIrqlId[]

14.3.8.6 const IRQn_Type g_flexcanBusOffIrqlId[]

14.3.8.7 const IRQn_Type g_flexcanOredMessageBufferIrqlId[]

Chapter 15

FlexTimer (FTM)

15.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the FlexTimer (FTM) block of Kinetis devices.

Modules

- [FlexTimer HAL driver](#)
- [FlexTimer Peripheral Driver](#)

FlexTimer HAL driver

15.2 FlexTimer HAL driver

15.2.1 Overview

This section describes the programming interface of the FlexTimer HAL driver.

Data Structures

- union `ftm_edge_mode_t`
FlexTimer edge mode. [More...](#)
- struct `ftm_pwm_param_t`
FlexTimer driver PWM parameter. [More...](#)
- struct `ftm_phase_params_t`
FlexTimer quadrature decode phase parameters. [More...](#)

Macros

- `#define HW_CHAN0 (0U)`
Channel number for CHAN0.
- `#define HW_CHAN1 (1U)`
Channel number for CHAN1.
- `#define HW_CHAN2 (2U)`
Channel number for CHAN2.
- `#define HW_CHAN3 (3U)`
Channel number for CHAN3.
- `#define HW_CHAN4 (4U)`
Channel number for CHAN4.
- `#define HW_CHAN5 (5U)`
Channel number for CHAN5.
- `#define HW_CHAN6 (6U)`
Channel number for CHAN6.
- `#define HW_CHAN7 (7U)`
Channel number for CHAN7.

Enumerations

- enum `ftm_clock_source_t`
FlexTimer clock source selection.
- enum `ftm_counting_mode_t`
FlexTimer counting mode selection.
- enum `ftm_clock_ps_t`
FlexTimer pre-scaler factor selection for the clock source.
- enum `ftm_deadtime_ps_t`
FlexTimer pre-scaler factor for the deadtime insertion.
- enum `ftm_config_mode_t`
FlexTimer operation mode, capture, output, dual.
- enum `ftm_input_capture_edge_mode_t`

- *FlexTimer input capture edge mode, rising edge, or falling edge.*
- enum **ftm_output_compare_edge_mode_t**
FlexTimer output compare edge mode.
- enum **ftm_pwm_edge_mode_t**
FlexTimer PWM output pulse mode, high-true or low-true on match up.
- enum **ftm_dual_capture_edge_mode_t**
FlexTimer dual capture edge mode, one shot or continuous.
- enum **ftm_quad_decode_mode_t**
FlexTimer quadrature decode modes, phase encode or count and direction mode.
- enum **ftm_quad_phase_polarity_t**
FlexTimer quadrature phase polarities, normal or inverted polarity.
- enum **ftm_sync_method_t**
FlexTimer sync options to update registers with buffer.
- enum **ftm_bdm_mode_t**
Options for the FlexTimer behaviour in BDM Mode.

Functions

- static void **FTM_HAL_SetClockSource** (uint32_t ftmBaseAddr, **ftm_clock_source_t** clock)
Sets the FTM clock source.
- static uint8_t **FTM_HAL_GetClockSource** (uint32_t ftmBaseAddr)
Reads the FTM clock source.
- static void **FTM_HAL_SetClockPs** (uint32_t ftmBaseAddr, **ftm_clock_ps_t** ps)
Sets the FTM clock divider.
- static uint8_t **FTM_HAL_GetClockPs** (uint32_t ftmBaseAddr)
Reads the FTM clock divider.
- static void **FTM_HAL_EnableTimerOverflowInt** (uint32_t ftmBaseAddr)
Enables the FTM peripheral timer overflow interrupt.
- static void **FTM_HAL_DisableTimerOverflowInt** (uint32_t ftmBaseAddr)
Disables the FTM peripheral timer overflow interrupt.
- static bool **FTM_HAL_IsOverflowIntEnabled** (uint32_t baseAddr)
Reads the bit that controls enabling the FTM timer overflow interrupt.
- static void **FTM_HAL_ClearTimerOverflow** (uint32_t ftmBaseAddr)
Clears the timer overflow interrupt flag.
- static bool **FTM_HAL_HasTimerOverflowed** (uint32_t ftmBaseAddr)
Returns the FTM peripheral timer overflow interrupt flag.
- static void **FTM_HAL_SetCpwms** (uint32_t ftmBaseAddr, uint8_t mode)
Sets the FTM center-aligned PWM select.
- static void **FTM_HAL_SetCounter** (uint32_t ftmBaseAddr, uint16_t val)
Sets the FTM peripheral current counter value.
- static uint16_t **FTM_HAL_GetCounter** (uint32_t ftmBaseAddr)
Returns the FTM peripheral current counter value.
- static void **FTM_HAL_SetMod** (uint32_t ftmBaseAddr, uint16_t val)
Sets the FTM peripheral timer modulo value.
- static uint16_t **FTM_HAL_GetMod** (uint32_t ftmBaseAddr)
Returns the FTM peripheral counter modulo value.
- static void **FTM_HAL_SetCounterInitVal** (uint32_t ftmBaseAddr, uint16_t val)
Sets the FTM peripheral timer counter initial value.
- static uint16_t **FTM_HAL_GetCounterInitVal** (uint32_t ftmBaseAddr)
Returns the FTM peripheral counter initial value.

FlexTimer HAL driver

- static void **FTM_HAL_SetChnMSnBAMode** (uint32_t ftmBaseAddr, uint8_t channel, uint8_t selection)
Sets the FTM peripheral timer channel mode.
- static void **FTM_HAL_SetChnEdgeLevel** (uint32_t ftmBaseAddr, uint8_t channel, uint8_t level)
Sets the FTM peripheral timer channel edge level.
- static uint8_t **FTM_HAL_GetChnMode** (uint32_t ftmBaseAddr, uint8_t channel)
Gets the FTM peripheral timer channel mode.
- static uint8_t **FTM_HAL_GetChnEdgeLevel** (uint32_t ftmBaseAddr, uint8_t channel)
Gets the FTM peripheral timer channel edge level.
- static void **FTM_HAL_SetChnDmaCmd** (uint32_t ftmBaseAddr, uint8_t channel, bool val)
Enables or disables the FTM peripheral timer channel DMA.
- static bool **FTM_HAL_IsChnDma** (uint32_t ftmBaseAddr, uint8_t channel)
Returns whether the FTM peripheral timer channel DMA is enabled.
- static void **FTM_HAL_EnableChnInt** (uint32_t ftmBaseAddr, uint8_t channel)
Enables the FTM peripheral timer channel(n) interrupt.
- static void **FTM_HAL_DisableChnInt** (uint32_t ftmBaseAddr, uint8_t channel)
Disables the FTM peripheral timer channel(n) interrupt.
- static bool **FTM_HAL_HasChnEventOccurred** (uint32_t ftmBaseAddr, uint8_t channel)
Returns whether any event for the FTM peripheral timer channel has occurred.
- static void **FTM_HAL_ClearChnEventFlag** (uint32_t ftmBaseAddr, uint8_t channel)
Clear the channel flag by writing a 0 to the CHF bit.
- static void **FTM_HAL_SetChnCountVal** (uint32_t ftmBaseAddr, uint8_t channel, uint16_t val)
Sets the FTM peripheral timer channel counter value.
- static uint16_t **FTM_HAL_GetChnCountVal** (uint32_t ftmBaseAddr, uint8_t channel)
Gets the FTM peripheral timer channel counter value.
- static uint32_t **FTM_HAL_GetChnEventStatus** (uint32_t ftmBaseAddr, uint8_t channel)
Gets the FTM peripheral timer channel event status.
- static void **FTM_HAL_ClearChnEventStatus** (uint32_t ftmBaseAddr, uint8_t channel)
Clears the FTM peripheral timer all channel event status.
- static void **FTM_HAL_SetOutmaskReg** (uint32_t ftmBaseAddr, uint32_t regVal)
Writes the provided value to the OUTMASK register.
- static void **FTM_HAL_SetChnOutputMask** (uint32_t ftmBaseAddr, uint8_t channel, bool mask)
Sets the FTM peripheral timer channel output mask.
- static void **FTM_HAL_SetChnOutputInitState** (uint32_t ftmBaseAddr, uint8_t channel, uint8_t state)
Sets the FTM peripheral timer channel output initial state 0 or 1.
- static void **FTM_HAL_SetChnOutputPolarity** (uint32_t ftmBaseAddr, uint8_t channel, uint8_t pol)
Sets the FTM peripheral timer channel output polarity.
- static void **FTM_HAL_SetChnFaultInputPolarity** (uint32_t ftmBaseAddr, uint8_t channel, uint8_t pol)
Sets the FTM peripheral timer channel input polarity.
- static void **FTM_HAL_EnableFaultInt** (uint32_t ftmBaseAddr)
Enables the FTM peripheral timer fault interrupt.
- static void **FTM_HAL_DisableFaultInt** (uint32_t ftmBaseAddr)
Disables the FTM peripheral timer fault interrupt.
- static void **FTM_HAL_SetFaultControlMode** (uint32_t ftmBaseAddr, uint8_t mode)
Defines the FTM fault control mode.
- static void **FTM_HAL_SetCaptureTestCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the FTM peripheral timer capture test mode.
- static void **FTM_HAL_SetWriteProtectionCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the FTM write protection.

- static void **FTM_HAL_Enable** (uint32_t *ftmBaseAddr*, bool *enable*)

Enables the FTM peripheral timer group.
- static void **FTM_HAL_SetInitChnOutputCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Initializes the channels output.
- static void **FTM_HAL_SetPwmSyncMode** (uint32_t *ftmBaseAddr*, bool *enable*)

Sets the FTM peripheral timer sync mode.
- static void **FTM_HAL_SetSoftwareTriggerCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Enables or disables the FTM peripheral timer software trigger.
- void **FTM_HAL_SetHardwareSyncTriggerSrc** (uint32_t *ftmBaseAddr*, uint32_t *trigger_num*, bool *enable*)

Sets the FTM peripheral timer hardware trigger.
- static void **FTM_HAL_SetOutmaskPwmSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Determines when the OUTMASK register is updated with the value of its buffer.
- static void **FTM_HAL_SetCountReinitSyncCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Determines if the FTM counter is re-initialized when the selected trigger for synchronization is detected.
- static void **FTM_HAL_SetMaxLoadingCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Enables or disables the FTM peripheral timer maximum loading points.
- static void **FTM_HAL_SetMinLoadingCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Enables or disables the FTM peripheral timer minimum loading points.
- uint32_t **FTM_HAL_GetChnPairIndex** (uint8_t *channel*)

Combines the channel control.
- static void **FTM_HAL_SetDualChnFaultCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables the FTM peripheral timer channel pair fault control.
- static void **FTM_HAL_SetDualChnPwmSyncCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables or disables the FTM peripheral timer channel pair counter PWM sync.
- static void **FTM_HAL_SetDualChnDeadtimeCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables or disabled the FTM peripheral timer channel pair deadtime insertion.
- static void **FTM_HAL_SetDualChnDecapCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables or disables the FTM peripheral timer channel dual edge capture decap.
- static void **FTM_HAL_SetDualEdgeCaptureCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables the FTM peripheral timer dual edge capture mode.
- static void **FTM_HAL_SetDualChnCompCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables or disables the FTM peripheral timer channel pair output complement mode.
- static void **FTM_HAL_SetDualChnCombineCmd** (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*)

Enables or disables the FTM peripheral timer channel pair output combine mode.
- static void **FTM_HAL_SetDeadtimePrescale** (uint32_t *ftmBaseAddr*, **ftm_deadtime_ps_t** *divider*)

Sets the FTM deadtime divider.
- static void **FTM_HAL_SetDeadtimeCount** (uint32_t *ftmBaseAddr*, uint8_t *count*)

Sets the FTM deadtime value.
- static void **FTM_HAL_SetInitTriggerCmd** (uint32_t *ftmBaseAddr*, bool *enable*)

Enables or disables the generation of the trigger when the FTM counter is equal to the CNTIN register.
- void **FTM_HAL_SetChnTriggerCmd** (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *val*)

Enables or disables the generation of the FTM peripheral timer channel trigger.

FlexTimer HAL driver

- static bool **FTM_HAL_IsChnTriggerGenerated** (uint32_t ftmBaseAddr)
Checks whether any channel trigger event has occurred.
- static uint8_t **FTM_HAL_GetDetectedFaultInput** (uint32_t ftmBaseAddr)
Gets the FTM detected fault input.
- static bool **FTM_HAL_IsWriteProtectionEnabled** (uint32_t ftmBaseAddr)
Checks whether the write protection is enabled.
- static void **FTM_HAL_SetQuadDecoderCmd** (uint32_t ftmBaseAddr, bool enable)
Enables the channel quadrature decoder.
- static void **FTM_HAL_SetQuadPhaseAFilterCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the phase A input filter.
- static void **FTM_HAL_SetQuadPhaseBFilterCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the phase B input filter.
- static void **FTM_HAL_SetQuadPhaseAPolarity** (uint32_t ftmBaseAddr, **ftm_quad_phase_polarity_t** mode)
Selects polarity for the quadrature decode phase A input.
- static void **FTM_HAL_SetQuadPhaseBPolarity** (uint32_t ftmBaseAddr, **ftm_quad_phase_polarity_t** mode)
Selects polarity for the quadrature decode phase B input.
- static void **FTM_HAL_SetQuadMode** (uint32_t ftmBaseAddr, **ftm_quad_decode_mode_t** quadMode)
Sets the encoding mode used in quadrature decoding mode.
- static uint8_t **FTM_HAL_GetQuadDir** (uint32_t ftmBaseAddr)
Gets the FTM counter direction in quadrature mode.
- static uint8_t **FTM_HAL_GetQuadTimerOverflowDir** (uint32_t ftmBaseAddr)
Gets the Timer overflow direction in quadrature mode.
- void **FTM_HAL_SetChnInputCaptureFilter** (uint32_t ftmBaseAddr, uint8_t channel, uint8_t val)
Sets the FTM peripheral timer channel input capture filter value.
- static void **FTM_HAL_SetFaultInputFilterVal** (uint32_t ftmBaseAddr, uint32_t val)
Sets the fault input filter value.
- static void **FTM_HAL_SetFaultInputFilterCmd** (uint32_t ftmBaseAddr, uint8_t inputNum, bool val)
Enables or disables the fault input filter.
- static void **FTM_HAL_SetFaultInputCmd** (uint32_t ftmBaseAddr, uint8_t inputNum, bool val)
Enables or disables the fault input.
- static void **FTM_HAL_SetDualChnInvertCmd** (uint32_t ftmBaseAddr, uint8_t chnlPairNum, bool val)
Enables or disables the channel invert for a channel pair.
- static void **FTM_HAL_SetInvctrlReg** (uint32_t ftmBaseAddr, uint32_t regVal)
Writes the provided value to the Inverting control register.
- static void **FTM_HAL_SetChnSoftwareCtrlCmd** (uint32_t ftmBaseAddr, uint8_t channel, bool val)
Enables or disables the channel software output control.
- static void **FTM_HAL_SetChnSoftwareCtrlVal** (uint32_t ftmBaseAddr, uint8_t channel, bool val)
Sets the channel software output control value.
- static void **FTM_HAL_SetPwmLoadCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the loading of MOD, CNTIN and CV with values of their write buffer.
- static void **FTM_HAL_SetPwmLoadChnSelCmd** (uint32_t ftmBaseAddr, uint8_t channel, bool val)
Includes or excludes the channel in the matching process.
- static void **FTM_HAL_SetGlobalTimeBaseOutputCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the FTM global time base signal generation to other FTM's.
- static void **FTM_HAL_SetGlobalTimeBaseCmd** (uint32_t ftmBaseAddr, bool enable)
Enables or disables the FTM timer global time base.

- static void **FTM_HAL_SetBdmMode** (uint32_t *ftmBaseAddr*, *ftm_bdm_mode_t* *val*)
Sets the BDM mode.
- static void **FTM_HAL_SetTofFreq** (uint32_t *ftmBaseAddr*, uint8_t *val*)
Sets the FTM timer TOF Frequency.
- void **FTM_HAL_SetSyncMode** (uint32_t *ftmBaseAddr*, uint32_t *syncMethod*)
Sets the FTM register synchronization method.
- static void **FTM_HAL_SetSwoctrlHardwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets the sync mode for the FTM SWOCTRL register when using a hardware trigger.
- static void **FTM_HAL_SetInvctrlHardwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM INVCTRL register when using a hardware trigger.
- static void **FTM_HAL_SetOutmaskHardwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM OUTMASK register when using a hardware trigger.
- static void **FTM_HAL_SetModCtinCvHardwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM MOD, CNTIN and CV registers when using a hardware trigger.
- static void **FTM_HAL_SetCounterHardwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM counter register when using a hardware trigger.
- static void **FTM_HAL_SetSwoctrlSoftwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM SWOCTRL register when using a software trigger.
- static void **FTM_HAL_SetInvctrlSoftwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM INVCTRL register when using a software trigger.
- static void **FTM_HAL_SetOutmaskSoftwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM OUTMASK register when using a software trigger.
- static void **FTM_HAL_SetModCtinCvSoftwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM MOD, CNTIN and CV registers when using a software trigger.
- static void **FTM_HAL_SetCounterSoftwareSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets sync mode for FTM counter register when using a software trigger.
- static void **FTM_HAL_SetPwmSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets the PWM synchronization mode to enhanced or legacy.
- static void **FTM_HAL_SetSwoctrlPwmSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets the SWOCTRL register PWM synchronization mode.
- static void **FTM_HAL_SetInvctrlPwmSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets the INVCTRL register PWM synchronization mode.
- static void **FTM_HAL_SetCtinPwmSyncModeCmd** (uint32_t *ftmBaseAddr*, bool *enable*)
Sets the CNTIN register PWM synchronization mode.
- void **FTM_HAL_Reset** (uint32_t *ftmBaseAddr*)
Resets the FTM registers.
- void **FTM_HAL_Init** (uint32_t *ftmBaseAddr*)
Initializes the FTM.
- void **FTM_HAL_EnablePwmMode** (uint32_t *ftmBaseAddr*, *ftm_pwm_param_t* **config*, uint8_t *channel*)
Enables the FTM timer when it is PWM output mode.
- void **FTM_HAL_DisablePwmMode** (uint32_t *ftmBaseAddr*, *ftm_pwm_param_t* **config*, uint8_t *channel*)
Disables the PWM output mode.

15.2.2 Data Structure Documentation

15.2.2.1 union ftm_edge_mode_t

15.2.2.2 struct ftm_pwm_param_t

Data Fields

- `ftm_config_mode_t mode`
FlexTimer PWM operation mode.
- `ftm_pwm_edge_mode_t edgeMode`
PWM output mode.
- `uint32_t uFrequencyHZ`
PWM period in Hz.
- `uint32_t uDutyCyclePercent`
PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...
- `uint16_t uFirstEdgeDelayPercent`
Used only in combined PWM mode to generate asymmetrical PWM.

15.2.2.2.0.31 Field Documentation

15.2.2.2.0.31.1 uint32_t ftm_pwm_param_t::uDutyCyclePercent

100=active signal (100% duty cycle).

15.2.2.2.0.31.2 uint16_t ftm_pwm_param_t::uFirstEdgeDelayPercent

Specifies the delay to the first edge in a PWM period. If unsure please leave as 0, should be specified as percentage of the PWM period

15.2.2.3 struct ftm_phase_params_t

Data Fields

- `bool kFtmPhaseInputFilter`
false: disable phase filter, true: enable phase filter
- `uint32_t kFtmPhaseFilterVal`
Filter value, used only if phase input filter is enabled.
- `ftm_quad_phase_polarity_t kFtmPhasePolarity`
kFtmQuadPhaseNormal or kFtmQuadPhaseInvert

15.2.3 Macro Definition Documentation

15.2.3.1 `#define HW_CHAN0 (0U)`

15.2.3.2 `#define HW_CHAN1 (1U)`

15.2.3.3 `#define HW_CHAN2 (2U)`

15.2.3.4 `#define HW_CHAN3 (3U)`

15.2.3.5 `#define HW_CHAN4 (4U)`

15.2.3.6 `#define HW_CHAN5 (5U)`

15.2.3.7 `#define HW_CHAN6 (6U)`

15.2.3.8 `#define HW_CHAN7 (7U)`

15.2.4 Enumeration Type Documentation

15.2.4.1 `enum ftm_output_compare_edge_mode_t`

Toggle, clear or set.

15.2.5 Function Documentation

15.2.5.1 `static void FTM_HAL_SetClockSource (uint32_t ftmBaseAddr, ftm_clock_source_t clock) [inline], [static]`

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>clock</i>	The FTM peripheral clock selection bits - 00: No clock 01: system clock 10: fixed clock 11: External clock

15.2.5.2 `static uint8_t FTM_HAL_GetClockSource (uint32_t ftmBaseAddr) [inline], [static]`

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Returns

The FTM clock source selection
bits - 00: No clock 01: system clock 10: fixed clock 11:External clock

15.2.5.3 static void FTM_HAL_SetClockPs (uint32_t *ftmBaseAddr*, *ftm_clock_ps_t ps*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>ps</i>	The FTM peripheral clock pre-scale divider

15.2.5.4 static uint8_t FTM_HAL_GetClockPs (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Returns

The FTM clock pre-scale divider

15.2.5.5 static void FTM_HAL_EnableTimerOverflowInt (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

15.2.5.6 static void FTM_HAL_DisableTimerOverflowInt (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

15.2.5.7 static bool FTM_HAL_IsOverflowIntEnabled (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	FTM module base address.
-----------------	--------------------------

Return values

<i>true</i>	if overflow interrupt is enabled, false if not
-------------	--

15.2.5.8 static void FTM_HAL_ClearTimerOverflow (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

15.2.5.9 static bool FTM_HAL_HasTimerOverflowed (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>true</i>	if overflow, false if not
-------------	---------------------------

15.2.5.10 static void FTM_HAL_SetCpwms (uint32_t *ftmBaseAddr*, uint8_t *mode*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>mode</i>	1:upcounting mode 0:up_down counting mode

15.2.5.11 static void FTM_HAL_SetCounter (uint32_t *ftmBaseAddr*, uint16_t *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>val</i>	FTM timer counter value to be set

15.2.5.12 static uint16_t FTM_HAL_GetCounter (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>current</i>	FTM timer counter value
----------------	-------------------------

15.2.5.13 static void FTM_HAL_SetMod (uint32_t *ftmBaseAddr*, uint16_t *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>val</i>	The value to be set to the timer modulo

15.2.5.14 static uint16_t FTM_HAL_GetMod (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>FTM</i>	timer modulo value
------------	--------------------

**15.2.5.15 static void FTM_HAL_SetCounterInitVal (uint32_t *ftmBaseAddr*, uint16_t *val*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>val</i>	initial value to be set

**15.2.5.16 static uint16_t FTM_HAL_GetCounterInitVal (uint32_t *ftmBaseAddr*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>FTM</i>	timer counter initial value
------------	-----------------------------

**15.2.5.17 static void FTM_HAL_SetChnMSnBAMode (uint32_t *ftmBaseAddr*, uint8_t
channel, uint8_t *selection*) [inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

FlexTimer HAL driver

<i>selection</i>	The mode to be set valid value MSnB:MSnA :00,01, 10, 11
------------------	---

15.2.5.18 static void FTM_HAL_SetChnEdgeLevel (uint32_t *ftmBaseAddr*, uint8_t *channel*, uint8_t *level*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>level</i>	The rising or falling edge to be set, valid value ELSnB:ELSnA :00,01, 10, 11

15.2.5.19 static uint8_t FTM_HAL_GetChnMode (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

Return values

<i>The</i>	MSnB:MSnA mode value, will be 00,01, 10, 11
------------	---

15.2.5.20 static uint8_t FTM_HAL_GetChnEdgeLevel (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

Return values

<i>The</i>	ELSnB:ELSnA mode value, will be 00,01, 10, 11
------------	---

15.2.5.21 static void FTM_HAL_SetChnDmaCmd (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>val</i>	enable or disable

**15.2.5.22 static bool FTM_HAL_IsChnDma (uint32_t *ftmBaseAddr*, uint8_t *channel*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

Return values

<i>true</i>	if enabled, false if disabled
-------------	-------------------------------

**15.2.5.23 static void FTM_HAL_EnableChnInt (uint32_t *ftmBaseAddr*, uint8_t *channel*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

**15.2.5.24 static void FTM_HAL_DisableChnInt (uint32_t *ftmBaseAddr*, uint8_t *channel*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

15.2.5.25 static bool FTM_HAL_HasChnEventOccurred (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

Return values

<i>true</i>	if event occurred, false otherwise.
-------------	-------------------------------------

15.2.5.26 static void FTM_HAL_ClearChnEventFlag (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

15.2.5.27 static void FTM_HAL_SetChnCountVal (uint32_t *ftmBaseAddr*, uint8_t *channel*, uint16_t *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>val</i>	counter value to be set

15.2.5.28 static uint16_t FTM_HAL_GetChnCountVal (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

Return values

<i>return</i>	channel counter value
---------------	-----------------------

15.2.5.29 static uint32_t FTM_HAL_GetChnEventStatus (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

Return values

<i>return</i>	channel event status value
---------------	----------------------------

15.2.5.30 static void FTM_HAL_ClearChnEventStatus (uint32_t *ftmBaseAddr*, uint8_t *channel*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number

15.2.5.31 static void FTM_HAL_SetOutmaskReg (uint32_t *ftmBaseAddr*, uint32_t *regVal*) [inline], [static]

This function will mask/uumask multiple channels.

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>regVal</i>	value to be written to the register

15.2.5.32 static void FTM_HAL_SetChnOutputMask (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *mask*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>mask</i>	mask to be set 0 or 1, unmasked or masked

15.2.5.33 static void FTM_HAL_SetChnOutputInitState (uint32_t *ftmBaseAddr*, uint8_t *channel*, uint8_t *state*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>state</i>	counter value to be set 0 or 1

15.2.5.34 static void FTM_HAL_SetChnOutputPolarity (uint32_t *ftmBaseAddr*, uint8_t *channel*, uint8_t *pol*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>pol</i>	polarity to be set 0 or 1

15.2.5.35 static void FTM_HAL_SetChnFaultInputPolarity (uint32_t *ftmBaseAddr*, uint8_t *channel*, uint8_t *pol*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number
<i>pol</i>	polarity to be set, 0: active high, 1:active low

15.2.5.36 static void FTM_HAL_EnableFaultInt (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

15.2.5.37 static void FTM_HAL_DisableFaultInt (uint32_t *ftmBaseAddr*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

15.2.5.38 static void FTM_HAL_SetFaultControlMode (uint32_t *ftmBaseAddr*, uint8_t *mode*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>mode,valid</i>	options are 1, 2, 3, 4

15.2.5.39 static void FTM_HAL_SetCaptureTestCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true to enable capture test mode, false to disable

15.2.5.40 static void FTM_HAL_SetWriteProtectionCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true: Write-protection is enabled, false: Write-protection is disabled

FlexTimer HAL driver

```
15.2.5.41 static void FTM_HAL_Enable ( uint32_t ftmBaseAddr, bool enable )  
    [inline], [static]
```

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true: all registers including FTM-specific registers are available false: only the TPM-compatible registers are available

15.2.5.42 static void FTM_HAL_SetInitChnOutputCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true: the channels output is initialized according to the state of OUTINIT reg false: has no effect

15.2.5.43 static void FTM_HAL_SetPwmSyncMode (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true: no restriction both software and hardware triggers can be used false: software trigger can only be used for MOD and CnV synch, hardware trigger only for OUTMASK and FTM counter synch.

15.2.5.44 static void FTM_HAL_SetSoftwareTriggerCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address.
<i>enable</i>	true: software trigger is selected, false: software trigger is not selected

15.2.5.45 void FTM_HAL_SetHardwareSyncTriggerSrc (uint32_t *ftmBaseAddr*, uint32_t *trigger_num*, bool *enable*)

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>trigger_num</i>	0, 1, 2 for trigger0, trigger1 and trigger3
<i>enable</i>	true: enable hardware trigger from field trigger_num for PWM synch false: disable hardware trigger from field trigger_num for PWM synch

15.2.5.46 static void FTM_HAL_SetOutmaskPwmSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true if OUTMASK register is updated only by PWM sync false if OUTMASK register is updated in all rising edges of the system clock

15.2.5.47 static void FTM_HAL_SetCountReinitSyncCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to update FTM counter when triggered , false to count normally

15.2.5.48 static void FTM_HAL_SetMaxLoadingCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to enable maximum loading point, false to disable

15.2.5.49 static void FTM_HAL_SetMinLoadingCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to enable minimum loading point, false to disable

15.2.5.50 **uint32_t FTM_HAL_GetChnPairIndex (uint8_t channel)**

Returns an index for each channel pair.

Parameters

<i>channel</i>	The FTM peripheral channel number.
----------------	------------------------------------

Returns

- 0 for channel pair 0 & 1
- 1 for channel pair 2 & 3
- 2 for channel pair 4 & 5
- 3 for channel pair 6 & 7

15.2.5.51 **static void FTM_HAL_SetDualChnFaultCmd (uint32_t ftmBaseAddr, uint8_t chnlPairNum, bool enable) [inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable fault control, false to disable

15.2.5.52 **static void FTM_HAL_SetDualChnPwmSyncCmd (uint32_t ftmBaseAddr, uint8_t chnlPairNum, bool enable) [inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable PWM synchronization, false to disable

FlexTimer HAL driver

15.2.5.53 **static void FTM_HAL_SetDualChnDeadtimeCmd (uint32_t *ftmBaseAddr*,
 uint8_t *chnlPairNum*, bool *enable*) [inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable deadtime insertion, false to disable

15.2.5.54 static void FTM_HAL_SetDualChnDecapCmd (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable dual edge capture mode, false to disable

15.2.5.55 static void FTM_HAL_SetDualEdgeCaptureCmd (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable dual edge capture, false to disable

15.2.5.56 static void FTM_HAL_SetDualChnCompCmd (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable complementary mode, false to disable

15.2.5.57 static void FTM_HAL_SetDualChnCombineCmd (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *enable*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>enable</i>	True to enable channel pair to combine, false to disable

15.2.5.58 static void FTM_HAL_SetDeadtimePrescale (uint32_t *ftmBaseAddr*, ftm_deadtime_ps_t *divider*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>divider</i>	The FTM peripheral prescale divider 0x :divided by 1, 10: divided by 4, 11:divided by 16

15.2.5.59 static void FTM_HAL_SetDeadtimeCount (uint32_t *ftmBaseAddr*, uint8_t *count*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>count</i>	The FTM peripheral prescale divider 0: no counts inserted, 1: 1 count is inserted, 2: 2 count is inserted....

15.2.5.60 static void FTM_HAL_SetInitTriggerCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to enable, false to disable

15.2.5.61 void FTM_HAL_SetChnTriggerCmd (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *val*)

Enables or disables the when the generation of the FTM peripheral timer channel trigger when the FTM counter is equal to its initial value. Channels 6 and 7 cannot be used as triggers.

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	Channel to be enabled, valid value 0, 1, 2, 3, 4, 5
<i>val</i>	True to enable, false to disable

**15.2.5.62 static bool FTM_HAL_IsChnTriggerGenerated (uint32_t *ftmBaseAddr*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>true</i>	if there is a channel trigger event, false if not.
-------------	--

**15.2.5.63 static uint8_t FTM_HAL_GetDetectedFaultInput (uint32_t *ftmBaseAddr*)
[inline], [static]**

This function reads the status for all fault inputs

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>Return</i>	fault byte
---------------	------------

**15.2.5.64 static bool FTM_HAL_IsWriteProtectionEnabled (uint32_t *ftmBaseAddr*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

FlexTimer HAL driver

Return values

<i>True</i>	if enabled, false if not
-------------	--------------------------

15.2.5.65 static void FTM_HAL_SetQuadDecoderCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to enable, false to disable

15.2.5.66 static void FTM_HAL_SetQuadPhaseAFilterCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true enables the phase input filter, false disables the filter

15.2.5.67 static void FTM_HAL_SetQuadPhaseBFilterCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true enables the phase input filter, false disables the filter

15.2.5.68 static void FTM_HAL_SetQuadPhaseAPolarity (uint32_t *ftmBaseAddr*, ftm_quad_phase_polarity_t *mode*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>mode</i>	0: Normal polarity, 1: Inverted polarity

15.2.5.69 **static void FTM_HAL_SetQuadPhaseBPolarity (uint32_t *ftmBaseAddr*,
 ftm_quad_phase_polarity_t *mode*) [inline], [static]**

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>mode</i>	0: Normal polarity, 1: Inverted polarity

**15.2.5.70 static void FTM_HAL_SetQuadMode (uint32_t *ftmBaseAddr*,
ftm_quad_decode_mode_t *quadMode*) [inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>quadMode</i>	0: Phase A and Phase B encoding mode 1: Count and direction encoding mode

**15.2.5.71 static uint8_t FTM_HAL_GetQuadDir (uint32_t *ftmBaseAddr*) [inline],
[static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>I</i>	if counting direction is increasing, 0 if counting direction is decreasing
----------	--

**15.2.5.72 static uint8_t FTM_HAL_GetQuadTimerOverflowDir (uint32_t *ftmBaseAddr*)
[inline], [static]**

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

Return values

<i>I</i>	if TOF bit was set on the top of counting, 0 if TOF bit was set on the bottom of counting
----------	--

15.2.5.73 **void FTM_HAL_SetChnInputCaptureFilter (uint32_t *ftmBaseAddr*, uint8_t *channel*, uint8_t *val*)**

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	The FTM peripheral channel number, only 0,1,2,3, channel 4, 5,6, 7 don't have.
<i>val</i>	Filter value to be set

15.2.5.74 static void FTM_HAL_SetFaultInputFilterVal (uint32_t *ftmBaseAddr*, uint32_t *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>val</i>	fault input filter value

15.2.5.75 static void FTM_HAL_SetFaultInputFilterCmd (uint32_t *ftmBaseAddr*, uint8_t *inputNum*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>inputNum</i>	fault input to be configured, valid value 0, 1, 2, 3
<i>val</i>	true to enable fault input filter, false to disable fault input filter

15.2.5.76 static void FTM_HAL_SetFaultInputCmd (uint32_t *ftmBaseAddr*, uint8_t *inputNum*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>inputNum</i>	fault input to be configured, valid value 0, 1, 2, 3
<i>val</i>	true to enable fault input, false to disable fault input

15.2.5.77 static void FTM_HAL_SetDualChnInvertCmd (uint32_t *ftmBaseAddr*, uint8_t *chnlPairNum*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>chnlPairNum</i>	The FTM peripheral channel pair number
<i>val</i>	true to enable channel inverting, false to disable channel inver

15.2.5.78 static void FTM_HAL_SetInvctrlReg (uint32_t *ftmBaseAddr*, uint32_t *regVal*) [inline], [static]

This function is enable/disable inverting control on multiple channel pairs.

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>regVal</i>	value to be written to the register

15.2.5.79 static void FTM_HAL_SetChnSoftwareCtrlCmd (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	Channel to be enabled or disabled
<i>val</i>	true to enable, channel output will be affected by software output control false to disable, channel output is unaffected

15.2.5.80 static void FTM_HAL_SetChnSoftwareCtrlVal (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address.
<i>channel</i>	Channel to be configured
<i>val</i>	True to set 1, false to set 0

15.2.5.81 static void FTM_HAL_SetPwmLoadCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true to enable, false to disable

15.2.5.82 static void FTM_HAL_SetPwmLoadChnSelCmd (uint32_t *ftmBaseAddr*, uint8_t *channel*, bool *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>channel</i>	Channel to be configured
<i>val</i>	true means include the channel in the matching process false means do not include channel in the matching process

15.2.5.83 static void FTM_HAL_SetGlobalTimeBaseOutputCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to enable, false to disable

15.2.5.84 static void FTM_HAL_SetGlobalTimeBaseCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	True to enable, false to disable

15.2.5.85 static void FTM_HAL_SetBdmMode (uint32_t *ftmBaseAddr*, *ftm_bdm_mode_t* *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>val</i>	FTM behaviour in BDM mode, options are defined in the enum <code>ftm_bdm_mode_t</code>

15.2.5.86 static void FTM_HAL_SetTofFreq (uint32_t *ftmBaseAddr*, uint8_t *val*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>val</i>	Value of the TOF bit set frequency

15.2.5.87 void FTM_HAL_SetSyncMode (uint32_t *ftmBaseAddr*, uint32_t *syncMethod*)

This function will set the necessary bits for the synchronization mode that user wishes to use.

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>syncMethod</i>	Synchronization method defined by <code>ftm_sync_method_t</code> enum. User can choose multiple synch methods by OR'ing options

15.2.5.88 static void FTM_HAL_SetSwoctrlHardwareSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means the hardware trigger activates register sync false means the hardware trigger does not activate register sync.

15.2.5.89 static void FTM_HAL_SetInvctrlHardwareSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means the hardware trigger activates register sync false means the hardware trigger does not activate register sync.

15.2.5.90 static void FTM_HAL_SetOutmaskHardwareSyncModeCmd (uint32_t ftmBaseAddr, bool enable) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means hardware trigger activates register sync false means hardware trigger does not activate register sync.

15.2.5.91 static void FTM_HAL_SetModCntinCvHardwareSyncModeCmd (uint32_t ftmBaseAddr, bool enable) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means hardware trigger activates register sync false means hardware trigger does not activate register sync.

15.2.5.92 static void FTM_HAL_SetCounterHardwareSyncModeCmd (uint32_t ftmBaseAddr, bool enable) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means hardware trigger activates register sync false means hardware trigger does not activate register sync.

15.2.5.93 static void FTM_HAL_SetSwoctrlSoftwareSyncModeCmd (uint32_t ftmBaseAddr, bool enable) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means software trigger activates register sync false means software trigger does not activate register sync.

15.2.5.94 static void FTM_HAL_SetInvctrlSoftwareSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means software trigger activates register sync false means software trigger does not activate register sync.

15.2.5.95 static void FTM_HAL_SetOutmaskSoftwareSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means software trigger activates register sync false means software trigger does not activate register sync.

15.2.5.96 static void FTM_HAL_SetModCntinCvSoftwareSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means software trigger activates register sync false means software trigger does not activate register sync.

15.2.5.97 static void FTM_HAL_SetCounterSoftwareSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

FlexTimer HAL driver

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means software trigger activates register sync false means software trigger does not activate register sync.

15.2.5.98 static void FTM_HAL_SetPwmSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means use Enhanced PWM synchronization false means to use Legacy mode

15.2.5.99 static void FTM_HAL_SetSwoctrlPwmSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means SWOCTRL register is updated by PWM synch false means SWOCTRL register is updated at all rising edges of system clock

15.2.5.100 static void FTM_HAL_SetInvctrlPwmSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means INVCTRL register is updated by PWM synch false means INVCTRL register is updated at all rising edges of system clock

15.2.5.101 static void FTM_HAL_SetCntinPwmSyncModeCmd (uint32_t *ftmBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>enable</i>	true means CNTIN register is updated by PWM synch false means CNTIN register is updated at all rising edges of system clock

15.2.5.102 void FTM_HAL_Reset (uint32_t *ftmBaseAddr*)

Parameters

<i>ftmBaseAddr</i>	The FTM base address
--------------------	----------------------

15.2.5.103 void FTM_HAL_Init (uint32_t *ftmBaseAddr*)

Parameters

<i>ftmBaseAddr</i>	The FTM base address.
--------------------	-----------------------

15.2.5.104 void FTM_HAL_EnablePwmMode (uint32_t *ftmBaseAddr*, ftm_pwm_param_t * *config*, uint8_t *channel*)

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>config</i>	PWM configuration parameter
<i>channel</i>	The channel or channel pair number(combined mode).

15.2.5.105 void FTM_HAL_DisablePwmMode (uint32_t *ftmBaseAddr*, ftm_pwm_param_t * *config*, uint8_t *channel*)

Parameters

<i>ftmBaseAddr</i>	The FTM base address
<i>config</i>	PWM configuration parameter
<i>channel</i>	The channel or channel pair number(combined mode).

FlexTimer Peripheral Driver

15.3 FlexTimer Peripheral Driver

15.3.1 Overview

This section describes the programming interface of the FlexTimer Peripheral driver.

15.3.2 FlexTimer Overview

The FlexTimer module is a timer that supports input capture, output compare, and generation of PWM signals. The current Kinetis SDK driver only supports the generation of PWM signals. The input capture and output compare will be supported in upcoming Kinetis SDK releases.

15.3.3 FlexTimer Initialization

1. To initialize the FlexTimer driver, call the [FTM_DRV_Init\(\)](#) function and pass the instance number of the FTM you want to use. For example, to use FTM0, pass a value of 0 to the initialization function.
2. Pass a user configuration structure [ftm_user_config_t](#), as shown here:

```
// FTM configuration structure
typedef struct FtmUserConfig {
    uint8_t toffFrequency;
    bool isFTMMode;
    uint8_t BDMMMode;
    bool isWriteProtection;

    bool isTimerOverFlowInterrupt;
    bool isFaultInterrupt;
} ftm_user_config_t;
```

15.3.4 FlexTimer Generate a PWM signal

FTM calls the [FTM_DRV_PwmStart\(\)](#) function to generate a PWM signal. Use this structure to configure different parameters for the PWM signal.

```
typedef struct FtmPwmParam
{
    ftm_config_mode_t mode;
    ftm_pwm_edge_mode_t edgeMode;
    uint32_t uFrequencyHZ;
    uint32_t uDutyCyclePercent;
                                0 = inactive signal(0% duty cycle)...
                                100 = active signal (100% duty cycle). //
    uint16_t uFirstEdgeDelayPercent;
                                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///
} ftm_pwm_param_t;
```

The mode options are kFtmEdgeAlignedPWM, kFtmCenterAlignedPWM, and kFtmCombinedPWM. For edge mode, the options available are kFtmHighTrue and kFtmLowTrue. Specify the PWM signal frequency in Hz and the duty cycle percentage (value between 0-100). If the PWM mode is kFtmCombinedPWM, and if the user chooses to specify a value for the uFirstEdgeDelayPercent, start of the PWM pulse will be delayed.

Data Structures

- struct `ftm_user_config_t`
Configuration structure that the user needs to set. [More...](#)

Functions

- void `FTM_DRV_Init` (uint8_t instance, `ftm_user_config_t` *info)
Initializes the FTM driver.
- void `FTM_DRV_Deinit` (uint8_t instance)
Shuts down the FTM driver.
- void `FTM_DRV_PwmStop` (uint8_t instance, `ftm_pwm_param_t` *param, uint8_t channel)
Stops channel PWM.
- void `FTM_DRV_PwmStart` (uint8_t instance, `ftm_pwm_param_t` *param, uint8_t channel)
Configures duty cycle and frequency and starts outputting PWM on specified channel.
- void `FTM_DRV_QuadDecodeStart` (uint8_t instance, `ftm_phase_params_t` *phaseAParams, `ftm_phase_params_t` *phaseBParams, `ftm_quad_decode_mode_t` quadMode)
Configures the parameters needed and activates quadrature decode mode.
- void `FTM_DRV_QuadDecodeStop` (uint8_t instance)
De-activates quadrature decode mode.
- void `FTM_DRV_CounterStart` (uint8_t instance, `ftm_counting_mode_t` countMode, uint32_t countStartVal, uint32_t countFinalVal, bool enableOverflowInt)
Starts the FTM counter.
- void `FTM_DRV_CounterStop` (uint8_t instance)
Stops the FTM counter.
- uint32_t `FTM_DRV_CounterRead` (uint8_t instance)
Reads back the current value of the FTM counter.
- void `FTM_DRV_SetTimeOverflowIntCmd` (uint32_t instance, bool overflowEnable)
Enables or disables the timer overflow interrupt.
- void `FTM_DRV_SetFaultIntCmd` (uint32_t instance, bool faultEnable)
Enables or disables the fault interrupt.
- void `FTM_DRV_IRQHandler` (uint8_t instance)
Action to take when an FTM interrupt is triggered.

Variables

- const uint32_t `g_ftmBaseAddr` []
Table of base addresses for FTM instances.
- const IRQn_Type `g_ftmIrqId` []
Table to save FTM IRQ enumeration numbers defined in the CMSIS header file.

15.3.5 Data Structure Documentation

15.3.5.1 struct `ftm_user_config_t`

Data Fields

- `uint8_t tofFrequency`
Select ratio between number of overflows to times TOF is set.
- `ftm_bdm_mode_t BDMMode`
Select FTM behavior in BDM mode.
- `bool isWriteProtection`
true: enable write protection, false: write protection is disabled
- `uint32_t syncMethod`
Register synch options available in the `ftm_sync_method_t` enumeration.

15.3.6 Function Documentation

15.3.6.1 void `FTM_DRV_Init` (`uint8_t instance`, `ftm_user_config_t * info`)

Parameters

<code>instance</code>	The FTM peripheral instance number.
<code>info</code>	The FTM user configuration structure.

15.3.6.2 void `FTM_DRV_Deinit` (`uint8_t instance`)

Parameters

<code>instance</code>	The FTM peripheral instance number.
-----------------------	-------------------------------------

15.3.6.3 void `FTM_DRV_PwmStop` (`uint8_t instance`, `ftm_pwm_param_t * param`, `uint8_t channel`)

Parameters

<code>instance</code>	The FTM peripheral instance number.
-----------------------	-------------------------------------

<i>param</i>	FTM driver PWM parameter to configure PWM options
<i>channel</i>	The channel number. In combined mode, the code will find the channel pair associated with the channel number passed in.

15.3.6.4 void FTM_DRV_PwmStart (uint8_t *instance*, ftm_pwm_param_t * *param*, uint8_t *channel*)

Parameters

<i>instance</i>	The FTM peripheral instance number.
<i>param</i>	FTM driver PWM parameter to configure PWM options
<i>channel</i>	The channel number. In combined mode, the code will find the channel pair associated with the channel number passed in.

15.3.6.5 void FTM_DRV_QuadDecodeStart (uint8_t *instance*, ftm_phase_params_t * *phaseAParams*, ftm_phase_params_t * *phaseBParams*, ftm_quad_decode_mode_t *quadMode*)

Parameters

<i>instance</i>	Instance number of the FTM module.
<i>phaseAParams</i>	Phase A configuration parameters
<i>phaseBParams</i>	Phase B configuration parameters
<i>quadMode</i>	Selects encoding mode used in quadrature decoder mode

15.3.6.6 void FTM_DRV_QuadDecodeStop (uint8_t *instance*)

Parameters

<i>instance</i>	Instance number of the FTM module.
-----------------	------------------------------------

15.3.6.7 void FTM_DRV_CounterStart (uint8_t *instance*, ftm_counting_mode_t *countMode*, uint32_t *countStartVal*, uint32_t *countFinalVal*, bool *enableOverflowInt*)

This function provides access to the FTM counter. The counter can be run in Up-counting and Up-down counting modes. To run the counter in Free running mode, choose Up-counting option and provide 0x0

FlexTimer Peripheral Driver

for the countStartVal and 0xFFFF for countFinalVal.

Parameters

<i>instance</i>	The FTM peripheral instance number.
<i>countMode</i>	The FTM counter mode defined by <code>ftm_counting_mode_t</code> .
<i>countStartVal</i>	The starting value that is stored in the CNTIN register.
<i>countFinalVal</i>	The final value that is stored in the MOD register.
<i>enable-OverflowInt</i>	true: enable timer overflow interrupt; false: disable

15.3.6.8 void FTM_DRV_CounterStop (`uint8_t instance`)

Parameters

<i>instance</i>	The FTM peripheral instance number.
-----------------	-------------------------------------

15.3.6.9 uint32_t FTM_DRV_CounterRead (`uint8_t instance`)

Parameters

<i>instance</i>	The FTM peripheral instance number.
-----------------	-------------------------------------

15.3.6.10 void FTM_DRV_SetTimeOverflowIntCmd (`uint32_t instance, bool overflowEnable`)

Parameters

<i>instance</i>	The FTM peripheral instance number.
<i>overflowEnable</i>	true: enable the timer overflow interrupt, false: disable

15.3.6.11 void FTM_DRV_SetFaultIntCmd (`uint32_t instance, bool faultEnable`)

Parameters

FlexTimer Peripheral Driver

<i>instance</i>	The FTM peripheral instance number.
<i>faultEnable</i>	true: enable the fault interrupt, false: disable

15.3.6.12 void FTM_DRV_IRQHandler (uint8_t *instance*)

The timer overflow flag is checked and cleared if set.

Parameters

<i>instance</i>	Instance number of the FTM module.
-----------------	------------------------------------

15.3.7 Variable Documentation

15.3.7.1 const uint32_t g_ftmBaseAddr[]

15.3.7.2 const IRQn_Type g_ftmIrqId[]

Chapter 16

General Purpose Input/Output (GPIO)

16.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the General Purpose Input/Output (GPIO) block of Kinetis devices.

Modules

- [GPIO HAL driver](#)
- [GPIO Peripheral driver](#)

GPIO HAL driver

16.2 GPIO HAL driver

16.2.1 Overview

This section describes the programming interface of the GPIO HAL driver.

Files

- file `fsl_gpio_hal.h`
GPIO hardware driver configuration.

Enumerations

- enum `gpio_pin_direction_t` {
 `kGpioDigitalInput` = 0,
 `kGpioDigitalOutput` = 1 }
GPIO direction definition.

Configuration

- void `GPIO_HAL_SetPinDir` (uint32_t baseAddr, uint32_t pin, `gpio_pin_direction_t` direction)
Sets the individual GPIO pin to general input or output.
- static void `GPIO_HAL_SetPortDir` (uint32_t baseAddr, uint32_t direction)
Sets the GPIO port pins to general input or output.

Status

- static `gpio_pin_direction_t` `GPIO_HAL_GetPinDir` (uint32_t baseAddr, uint32_t pin)
Gets the current direction of the individual GPIO pin.
- static uint32_t `GPIO_HAL_GetPortDir` (uint32_t baseAddr)
Gets the GPIO port pins direction.

Output Operation

- void `GPIO_HAL_WritePinOutput` (uint32_t baseAddr, uint32_t pin, uint32_t output)
Sets the output level of the individual GPIO pin to logic 1 or 0.
- static uint32_t `GPIO_HAL_ReadPinOutput` (uint32_t baseAddr, uint32_t pin)
Reads the current pin output.
- static void `GPIO_HAL_SetPinOutput` (uint32_t baseAddr, uint32_t pin)
Sets the output level of the individual GPIO pin to logic 1.
- static void `GPIO_HAL_ClearPinOutput` (uint32_t baseAddr, uint32_t pin)
Clears the output level of the individual GPIO pin to logic 0.
- static void `GPIO_HAL_TogglePinOutput` (uint32_t baseAddr, uint32_t pin)
Reverses the current output logic of the individual GPIO pin.

- static void **GPIO_HAL_WritePortOutput** (uint32_t baseAddr, uint32_t portOutput)
Sets the output of the GPIO port to a specific logic value.
- static uint32_t **GPIO_HAL_ReadPortOutput** (uint32_t baseAddr)
Reads out all pin output status of the current port.

Input Operation

- static uint32_t **GPIO_HAL_ReadPinInput** (uint32_t baseAddr, uint32_t pin)
Reads the current input value of the individual GPIO pin.
- static uint32_t **GPIO_HAL_ReadPortInput** (uint32_t baseAddr)
Reads the current input value of a specific GPIO port.

16.2.2 Enumeration Type Documentation

16.2.2.1 enum gpio_pin_direction_t

Enumerator

kGpioDigitalInput Set current pin as digital input.

kGpioDigitalOutput Set current pin as digital output.

16.2.3 Function Documentation

16.2.3.1 void **GPIO_HAL_SetPinDir** (*uint32_t baseAddr, uint32_t pin, gpio_pin_direction_t direction*)

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number
<i>direction</i>	GPIO directions <ul style="list-style-type: none"> • kGpioDigitalInput: set to input • kGpioDigitalOutput: set to output

16.2.3.2 static void **GPIO_HAL_SetPortDir** (*uint32_t baseAddr, uint32_t direction*) *[inline], [static]*

This function operates all 32 port pins.

GPIO HAL driver

Parameters

<i>baseAddr</i>	GPIO base address (PTA, PTB, PTC, etc.)
<i>direction</i>	GPIO directions <ul style="list-style-type: none">• 0: set to input• 1: set to output• LSB: pin 0• MSB: pin 31

16.2.3.3 static gpio_pin_direction_t GPIO_HAL_GetPinDir (uint32_t *baseAddr*, uint32_t *pin*) [inline], [static]

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number

Returns

GPIO directions

- kGpioDigitalInput: corresponding pin is set to input.
- kGpioDigitalOutput: corresponding pin is set to output.

16.2.3.4 static uint32_t GPIO_HAL_GetPortDir (uint32_t *baseAddr*) [inline], [static]

This function gets all 32-pin directions as a 32-bit integer.

Parameters

<i>baseAddr</i>	GPIO base address (PTA, PTB, PTC, etc.)
-----------------	---

Returns

GPIO directions. Each bit represents one pin. For each bit:

- 0: corresponding pin is set to input
- 1: corresponding pin is set to output
- LSB: pin 0
- MSB: pin 31

16.2.3.5 void GPIO_HAL_WritePinOutput (uint32_t *baseAddr*, uint32_t *pin*, uint32_t *output*)

GPIO HAL driver

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number
<i>output</i>	pin output logic level

**16.2.3.6 static uint32_t GPIO_HAL_ReadPinOutput (uint32_t *baseAddr*, uint32_t *pin*)
[inline], [static]**

Parameters

<i>baseAddr</i>	GPIO base address (PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number

Returns

current pin output status. 0 - Low logic, 1 - High logic

**16.2.3.7 static void GPIO_HAL_SetPinOutput (uint32_t *baseAddr*, uint32_t *pin*)
[inline], [static]**

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number

**16.2.3.8 static void GPIO_HAL_ClearPinOutput (uint32_t *baseAddr*, uint32_t *pin*)
[inline], [static]**

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number

**16.2.3.9 static void GPIO_HAL_TogglePinOutput (uint32_t *baseAddr*, uint32_t *pin*)
[inline], [static]**

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number

16.2.3.10 static void GPIO_HAL_WritePortOutput (uint32_t *baseAddr*, uint32_t *portOutput*) [inline], [static]

This function operates all 32 port pins.

Parameters

<i>baseAddr</i>	GPIO base address (PTA, PTB, PTC, etc.)
<i>portOutput</i>	<p>data to configure the GPIO output. Each bit represents one pin. For each bit:</p> <ul style="list-style-type: none"> • 0: set logic level 0 to pin • 1: set logic level 1 to pin • LSB: pin 0 • MSB: pin 31

16.2.3.11 static uint32_t GPIO_HAL_ReadPortOutput (uint32_t *baseAddr*) [inline], [static]

This function operates all 32 port pins.

Parameters

<i>baseAddr</i>	GPIO base address (PTA, PTB, PTC, etc.)
-----------------	---

Returns

current port output status. Each bit represents one pin. For each bit:

- 0: corresponding pin is outputting logic level 0
- 1: corresponding pin is outputting logic level 1
- LSB: pin 0
- MSB: pin 31

16.2.3.12 static uint32_t GPIO_HAL_ReadPinInput (uint32_t *baseAddr*, uint32_t *pin*) [inline], [static]

GPIO HAL driver

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
<i>pin</i>	GPIO port pin number

Returns

GPIO port input value

- 0: Pin logic level is 0, or is not configured for use by digital function.
- 1: Pin logic level is 1

16.2.3.13 **static uint32_t GPIO_HAL_ReadPortInput(uint32_t *baseAddr*) [inline], [static]**

This function gets all 32-pin input as a 32-bit integer.

Parameters

<i>baseAddr</i>	GPIO base address(PTA, PTB, PTC, etc.)
-----------------	--

Returns

GPIO port input data. Each bit represents one pin. For each bit:

- 0: Pin logic level is 0, or is not configured for use by digital function.
- 1: Pin logic level is 1.
- LSB: pin 0
- MSB: pin 31

16.3 GPIO Peripheral driver

16.3.1 Overview

This section describes the programming interface of the GPIO Peripheral driver. The GPIO Peripheral driver configures pins to digital input/output and provides API functions for input/output operations.

16.3.2 GPIO Pin Definitions

Define GPIO pins according to the target board configuration and ensure that the definitions are correct. Define a header file to store the GPIO pin names and the input/output configuration arrays defined in source files.

Include this header file in source files where you want to use GPIO driver to operate GPIO pins.

GPIO pin header file example:

```
// Feel free to change the pin names as what you want
enum _gpio_pins
{
    kGpioLED1 = GPIO_MAKE_PIN(HW_PORTC, 0x0),
    kGpioLED2 = GPIO_MAKE_PIN(HW_PORTC, 0x1),
    kGpioLED3 = GPIO_MAKE_PIN(HW_PORTC, 0x2),
    kGpioLED4 = GPIO_MAKE_PIN(HW_PORTC, 0x3),
};

// Extern input/output arrays defined in source files.
extern gpio_input_pin_t inputPin[];
extern gpio_output_pin_t outputPin[];
```

16.3.3 GPIO Initialization

1. To initialize the GPIO driver, define two arrays, the `gpio_input_pin_t` `inputPin[]` and the `gpio_output_pin_t` `outputPin[]`.
2. Then, call the `GPIO_DRV_Init()` function and pass these two arrays.

Example of the `inputPin` and `outputPin` array definition:

```
#include "gpio/fsl_gpio_driver.h"
#include "gpio_pin_header_file.h"

// Configure kGpioPTA2 as a digital input and enable the interrupt on the rising edge.
gpio_input_pin_t inputPin[] = {
{
    .pinName = kGpioPTA2,
    .config.isPullEnable = false,
    .config.pullSelect = kPortPullDown,
    .config.isPassiveFilterEnabled = false,
    .config.interrupt = kPortIntRisingEdge,
},
{
    //Note: This pinName must be defined here to indicate end of array.
    .pinName = GPIO_PINS_OUT_OF_RANGE,
}
```

GPIO Peripheral driver

```
};

// Configure the kGpioLED4 and kGpioPTB9 as a digital output and enable the high drive strength.
gpio_output_pin_t outputPin[] = {
{
    .pinName = kGpioLED4,
    .config.outputLogic = 0,
    .config.slewRate = kPortFastSlewRate,
    .config.driveStrength = kPortHighDriveStrength,
    .config.interrupt = kPortIntDisabled,
},
{
    .pinName = kGpioPTB9,
    .config.outputLogic = 0,
    .config.slewRate = kPortFastSlewRate,
    .config.driveStrength = kPortHighDriveStrength,
    .config.interrupt = kPortIntDisabled,
},
{
    //Note: This pinName must be defined here to indicate the end of array.
    .pinName = GPIO_PINS_OUT_OF_RANGE,
}
};

// Initializes GPIO pins.
GPIO_DRV_Init(inputPin, outputPin);
```

<note> If the digital filter is enabled, the digital filter clock source and width must be configured before calling the GPIO_DRV_Init function. To configure the digital filter, use this API function from porting the HAL driver. Each pin in the same port shares the same digital filter settings.</note>

```
void PORT_HAL_SetDigitalFilterClock(uint32_t baseAddr, port_digital_filter_clock_source_t clockSource);
void PORT_HAL_SetDigitalFilterWidth(uint32_t baseAddr, uint8_t width);
```

16.3.4 Output Operations

To use the output operation, configure the target GPIO pin as a digital output in the GPIO_DRV_Init function. The output operations include set, clear, and toggle of the output logic level. Three API functions are provided for these operations:

```
void GPIO_DRV_SetPinOutput(uint32_t pinName);
void GPIO_DRV_ClearPinOutput(uint32_t pinName);
void GPIO_DRV_TogglePinOutput(uint32_t pinName);
```

These functions are used when the logic level of the GPIO output is known.

Otherwise, a different function is provided to configure the output logic level according to a passed parameter:

```
void GPIO_DRV_WritePinOutput(uint32_t pinName, uint32_t output);
```

Use this function to change the GPIO output according to the results of other code. Pass an integer as the "uint32_t output" parameter. If that integer is not 0, it generates the high logic. If the integer is 0, it generates the low logic.

16.3.5 Input Operations

To use the input operation, configure the target GPIO pin as a digital input in the GPIO_DRV_Init function. For the input operation, this is the most commonly used API function:

```
uint32_t GPIO_DRV_ReadPinInput(uint32_t pinName);
```

This function returns the logic level read from a specific GPIO pin.

If the digital filter is enabled, use this function to disable it:

```
void GPIO_DRV_SetDigitalFilterCmd(uint32_t pinName, bool isDigitalFilterEnabled);
```

16.3.6 GPIO Interrupt

Enable a specific pin interrupt in GPIO initialization structures. Both output and input can trigger an interrupt. This API function clears the interrupt flag inside the interrupt handler:

```
void GPIO_DRV_ClearPinIntFlag(uint32_t pinName);
```

Files

- file [fsl_gpio_driver.h](#)

The GPIO driver uses the virtual GPIO name rather than an actual port and a pin number.

Data Structures

- struct [gpio_input_pin_t](#)
The GPIO input pin configuration structure. [More...](#)
- struct [gpio_output_pin_t](#)
The GPIO output pin configuration structure. [More...](#)
- struct [gpio_input_pin_user_config_t](#)
The GPIO input pin structure. [More...](#)
- struct [gpio_output_pin_user_config_t](#)
The GPIO output pin structure. [More...](#)

Variables

- const uint32_t [g_gpioBaseAddr](#) [HW_GPIO_INSTANCE_COUNT]
Table of base addresses for GPIO instances.
- const uint32_t [g_portBaseAddr](#) [HW_PORT_INSTANCE_COUNT]
Table of base addresses for PORT instances.

GPIO Peripheral driver

GPIO Pin Macros

- `#define GPIO_PINS_OUT_OF_RANGE (0xFFFFFFFFU)`
Indicates the end of a pin configuration structure.
- `#define GPIO_PORT_SHIFT (0x8U)`
Bits shifted for the GPIO port number.
- `#define GPIO_MAKE_PIN(r, p) (((r)<< GPIO_PORT_SHIFT) | (p))`
Combines the port number and the pin number into a single scalar value.
- `#define GPIO_EXTRACT_PORT(v) (((v) >> GPIO_PORT_SHIFT) & 0xFFU)`
Extracts the port number from a combined port and pin value.
- `#define GPIO_EXTRACT_PIN(v) ((v) & 0xFFU)`
Extracts the pin number from a combined port and pin value.

Initialization

- `void GPIO_DRV_Init (const gpio_input_pin_user_config_t *inputPins, const gpio_output_pin_user_config_t *outputPins)`
Initializes all GPIO pins used by the board.
- `void GPIO_DRV_InputPinInit (const gpio_input_pin_user_config_t *inputPin)`
Initializes one GPIO input pin used by the board.
- `void GPIO_DRV_OutputPinInit (const gpio_output_pin_user_config_t *outputPin)`
Initializes one GPIO output pin used by the board.

Pin Direction

- `gpio_pin_direction_t GPIO_DRV_GetPinDir (uint32_t pinName)`
Gets the current direction of the individual GPIO pin.
- `void GPIO_DRV_SetPinDir (uint32_t pinName, gpio_pin_direction_t direction)`
Sets the current direction of the individual GPIO pin.

Output Operations

- `void GPIO_DRV_WritePinOutput (uint32_t pinName, uint32_t output)`
Sets the output level of the individual GPIO pin to the logic 1 or 0.
- `void GPIO_DRV_SetPinOutput (uint32_t pinName)`
Sets the output level of the individual GPIO pin to the logic 1.
- `void GPIO_DRV_ClearPinOutput (uint32_t pinName)`
Sets the output level of the individual GPIO pin to the logic 0.
- `void GPIO_DRV_TogglePinOutput (uint32_t pinName)`
Reverses current output logic of the individual GPIO pin.

Input Operations

- `uint32_t GPIO_DRV_ReadPinInput (uint32_t pinName)`
Reads the current input value of the individual GPIO pin.

Interrupt

- void [GPIO_DRV_ClearPinIntFlag](#) (uint32_t pinName)
Clears the individual GPIO pin interrupt status flag.

16.3.7 Data Structure Documentation

16.3.7.1 struct gpio_input_pin_t

Although every pin is configurable, valid configurations depend on a specific device. Users should check the related reference manual to ensure that the specific feature is valid in an individual pin. A configuration of unavailable features is harmless, but takes no effect.

16.3.7.2 struct gpio_output_pin_t

Although every pin is configurable, valid configurations depend on a specific device. Users should check the related reference manual to ensure that the specific feature is valid in an individual pin. The configuration of unavailable features is harmless, but takes no effect.

Data Fields

- uint32_t outputLogic
Set default output logic.

16.3.7.2.0.32 Field Documentation

16.3.7.2.0.32.1 uint32_t gpio_output_pin_t::outputLogic

16.3.7.3 struct gpio_input_pin_user_config_t

Although the pinName is defined as a uint32_t type, values assigned to the pinName should be the enumeration names defined in the enum _gpio_pins.

Data Fields

- uint32_t pinName
Virtual pin name from enumeration defined by the user.
- [gpio_input_pin_t config](#)
Input pin configuration structure.

GPIO Peripheral driver

16.3.7.3.0.33 Field Documentation

16.3.7.3.0.33.1 `uint32_t gpio_input_pin_user_config_t::pinName`

16.3.7.3.0.33.2 `gpio_input_pin_t gpio_input_pin_user_config_t::config`

16.3.7.4 `struct gpio_output_pin_user_config_t`

Although the pinName is defined as a `uint32_t` type, values assigned to the pinName should be the enumeration names defined in the enum `_gpio_pins`.

Data Fields

- `uint32_t pinName`
Virtual pin name from enumeration defined by the user.
- `gpio_output_pin_t config`
Input pin configuration structure.

16.3.7.4.0.34 Field Documentation

16.3.7.4.0.34.1 `uint32_t gpio_output_pin_user_config_t::pinName`

16.3.7.4.0.34.2 `gpio_output_pin_t gpio_output_pin_user_config_t::config`

16.3.8 Macro Definition Documentation

16.3.8.1 `#define GPIO_PINS_OUT_OF_RANGE (0xFFFFFFFFU)`

16.3.8.2 `#define GPIO_PORT_SHIFT (0x8U)`

16.3.8.3 `#define GPIO_MAKE_PIN(r, p) (((r)<< GPIO_PORT_SHIFT) | (p))`

16.3.8.4 `#define GPIO_EXTRACT_PORT(v) (((v) >> GPIO_PORT_SHIFT) & 0xFFU)`

16.3.8.5 `#define GPIO_EXTRACT_PIN(v) ((v) & 0xFFU)`

16.3.9 Function Documentation

16.3.9.1 `void GPIO_DRV_Init (const gpio_input_pin_user_config_t * inputPins, const gpio_output_pin_user_config_t * outputPins)`

To initialize the GPIO driver, define two arrays similar to the `gpio_input_pin_user_config_t` `inputPin[]` array and the `gpio_output_pin_user_config_t` `outputPin[]` array in the user file. Then, call the `GPIO_DRV_Init()` function and pass in the two arrays. If the input or output pins are not needed, pass in a NULL.

This is an example to define an input pin array:

```
// Configure the kGpioPTA2 as digital input.
gpio_input_pin_user_config_t inputPin[] = {
{
    .pinName = kGpioPTA2,
    .config.isPullEnable = false,
    .config.pullSelect = kPortPullDown,
    .config.isPassiveFilterEnabled = false,
    .config.interrupt = kPortIntDisabled,
},
{
    // Note: This pinName must be defined here to indicate the end of the array.
    .pinName = GPIO_PINS_OUT_OF_RANGE,
}
};
```

Parameters

<i>inputPins</i>	input GPIO pins pointer.
<i>outputPins</i>	output GPIO pins pointer.

16.3.9.2 void GPIO_DRV_InputPinInit (const gpio_input_pin_user_config_t * *inputPin*)

Parameters

<i>inputPin</i>	input GPIO pins pointer.
-----------------	--------------------------

16.3.9.3 void GPIO_DRV_OutputPinInit (const gpio_output_pin_user_config_t * *outputPin*)

Parameters

<i>outputPin</i>	output GPIO pins pointer.
------------------	---------------------------

16.3.9.4 gpio_pin_direction_t GPIO_DRV_GetPinDir (uint32_t *pinName*)

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
----------------	---

Returns

GPIO directions.

- kGpioDigitalInput: corresponding pin is set as digital input.
- kGpioDigitalOutput: corresponding pin is set as digital output.

16.3.9.5 void GPIO_DRV_SetPinDir (uint32_t *pinName*, gpio_pin_direction_t *direction*)

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
<i>direction</i>	GPIO directions. <ul style="list-style-type: none"> • kGpioDigitalInput: corresponding pin is set as digital input. • kGpioDigitalOutput: corresponding pin is set as digital output.

16.3.9.6 void GPIO_DRV_WritePinOutput (uint32_t *pinName*, uint32_t *output*)

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
<i>output</i>	pin output logic level. <ul style="list-style-type: none"> • 0: corresponding pin output low logic level. • Non-0: corresponding pin output high logic level.

16.3.9.7 void GPIO_DRV_SetPinOutput (uint32_t *pinName*)

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
----------------	---

16.3.9.8 void GPIO_DRV_ClearPinOutput (uint32_t *pinName*)

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
----------------	---

16.3.9.9 void GPIO_DRV_TogglePinOutput (uint32_t *pinName*)

Parameters

GPIO Peripheral driver

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
----------------	---

16.3.9.10 `uint32_t GPIO_DRV_ReadPinInput (uint32_t pinName)`

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
----------------	---

Returns

GPIO port input value.

- 0: Pin logic level is 0, or is not configured for use by digital function.
- 1: Pin logic level is 1.

16.3.9.11 `void GPIO_DRV_ClearPinIntFlag (uint32_t pinName)`

Parameters

<i>pinName</i>	GPIO pin name defined by the user in the GPIO pin enumeration list.
----------------	---

16.3.10 Variable Documentation

16.3.10.1 `const uint32_t g_gpioBaseAddr[HW_GPIO_INSTANCE_COUNT]`

16.3.10.2 `const uint32_t g_portBaseAddr[HW_PORT_INSTANCE_COUNT]`

Chapter 17

Inter-Integrated Circuit (I2C)

17.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Inter-Integrated Circuit (I2C) block of Kinetis devices.

Modules

- I2C HAL driver
- I2C Master peripheral
- I2C Slave peripheral driver

17.2 I2C HAL driver

17.2.1 Overview

This section describes the programming interface of the I2C HAL driver.

17.2.2 I2C ICR Table

ICR (hex)	SCL divider	SDA hold value	SCL start hold value	SCL stop hold value
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21

17.2.3 I2C Clock rate formulas

I2C baud rate = bus_clock_Hz / (mult * SCL_divider)

SDA hold time = bus_clock_period_s * mult * SDA_hold_value

SCL start hold time = bus_clock_period_s * mult * SCL_start_hold_value

SCL stop hold time = bus_clock_period_s * mult * SCL_stop_hold_value

Enumerations

- enum `i2c_status_t` {

kStatus_I2C_Success = 0x0U,
 kStatus_I2C_Initialized = 0x1U,
 kStatus_I2C_Fail = 0x2U,
 kStatus_I2C_Busy = 0x3U,
 kStatus_I2C_Timeout = 0x4U,
 kStatus_I2C_ReceivedNak = 0x5U,
 kStatus_I2C_SlaveTxUnderrun = 0x6U,
 kStatus_I2C_SlaveRxOverrun = 0x7U,
 kStatus_I2C_ArbitrationLost = 0x8U,
 kStatus_I2C_StopSignalFail = 0x9U,
 kStatus_I2C_Idle = 0xAU,
 kStatus_I2C_NoReceiveInProgress = 0xBU,
 kStatus_I2C_NoSendInProgress = 0xCU }

I2C status return codes.

- enum `i2c_status_flag_t`
I2C status flags.
- enum `i2c_direction_t` {
kI2CReceive = 0U,
kI2CSend = 1U }

Direction of master and slave transfers.

Module controls

- void `I2C_HAL_Init` (uint32_t baseAddr)
Restores the I2C peripheral to reset state.
- static void `I2C_HAL_Enable` (uint32_t baseAddr)
Enables the I2C module operation.
- static void `I2C_HAL_Disable` (uint32_t baseAddr)
Disables the I2C module operation.

Pin functions

- static void `I2C_HAL_SetGlitchWidth` (uint32_t baseAddr, uint8_t glitchWidth)
Controls the width of the programmable glitch filter.

Low power

- static void `I2C_HAL_SetWakeupCmd` (uint32_t baseAddr, bool enable)
Controls the I2C wakeup enable.

I2C HAL driver

Baud rate

- void [I2C_HAL_SetBaudRate](#) (uint32_t baseAddr, uint32_t sourceClockInHz, uint32_t kbps, uint32_t *absoluteError_Hz)
Sets the I2C bus frequency for master transactions.
- static void [I2C_HAL_SetFreqDiv](#) (uint32_t baseAddr, uint8_t mult, uint8_t icr)
Sets the I2C baud rate multiplier and table entry.
- static void [I2C_HAL_SetSlaveBaudCtrlCmd](#) (uint32_t baseAddr, bool enable)
Slave baud rate control.

Bus operations

- void [I2C_HAL_SendStart](#) (uint32_t baseAddr)
Sends a START or a Repeated START signal on the I2C bus.
- [i2c_status_t I2C_HAL_SendStop](#) (uint32_t baseAddr)
Sends a STOP signal on the I2C bus.
- static void [I2C_HAL_SendAck](#) (uint32_t baseAddr)
Causes an ACK to be sent on the bus.
- static void [I2C_HAL_SendNak](#) (uint32_t baseAddr)
Causes a NAK to be sent on the bus.
- static void [I2C_HAL_SetDirMode](#) (uint32_t baseAddr, [i2c_direction_t](#) direction)
Selects either transmit or receive mode.
- static [i2c_direction_t I2C_HAL_GetDirMode](#) (uint32_t baseAddr)
Returns the currently selected transmit or receive mode.

Data transfer

- static uint8_t [I2C_HAL_ReadByte](#) (uint32_t baseAddr)
Returns the last byte of data read from the bus and initiate another read.
- static void [I2C_HAL_WriteByte](#) (uint32_t baseAddr, uint8_t byte)
Writes one byte of data to the I2C bus.
- uint8_t [I2C_HAL_ReadByteBlocking](#) (uint32_t baseAddr)
Returns the last byte of data read from the bus and initiate another read.
- bool [I2C_HAL_WriteByteBlocking](#) (uint32_t baseAddr, uint8_t byte)
Writes one byte of data to the I2C bus and wait till that byte is transferred successfully.
- [i2c_status_t I2C_HAL_MasterReceiveDataPolling](#) (uint32_t baseAddr, uint16_t slaveAddr, const uint8_t *cmdBuff, uint32_t cmdSize, uint8_t *rxBuff, uint32_t rxSize)
Performs a polling receive transaction on the I2C bus.
- [i2c_status_t I2C_HAL_MasterSendDataPolling](#) (uint32_t baseAddr, uint16_t slaveAddr, const uint8_t *cmdBuff, uint32_t cmdSize, const uint8_t *txBuff, uint32_t txSize)
Performs a polling send transaction on the I2C bus.
- [i2c_status_t I2C_HAL_SlaveSendDataPolling](#) (uint32_t baseAddr, const uint8_t *txBuff, uint32_t txSize)
Send out multiple bytes of data using polling method.
- [i2c_status_t I2C_HAL_SlaveReceiveDataPolling](#) (uint32_t baseAddr, uint8_t *rxBuff, uint32_t rxSize)
Receive multiple bytes of data using polling method.

Slave address

- void [I2C_HAL_SetAddress7bit](#) (uint32_t baseAddr, uint8_t address)
Sets the primary 7-bit slave address.
- void [I2C_HAL_SetAddress10bit](#) (uint32_t baseAddr, uint16_t address)
Sets the primary slave address and enables 10-bit address mode.
- static void [I2C_HAL_SetExtensionAddrCmd](#) (uint32_t baseAddr, bool enable)
Enables or disables the extension address (10-bit).
- static bool [I2C_HAL_GetExtensionAddrCmd](#) (uint32_t baseAddr)
Returns whether the extension address is enabled or not.
- static void [I2C_HAL_SetGeneralCallCmd](#) (uint32_t baseAddr, bool enable)
Controls whether the general call address is recognized.
- static void [I2C_HAL_SetRangeMatchCmd](#) (uint32_t baseAddr, bool enable)
Enables or disables the slave address range matching.
- static void [I2C_HAL_SetUpperAddress7bit](#) (uint32_t baseAddr, uint8_t address)
Sets the upper slave address.

Status

- static bool [I2C_HAL_GetStatusFlag](#) (uint32_t baseAddr, [i2c_status_flag_t](#) statusFlag)
Gets the I2C status flag state.
- static bool [I2C_HAL_IsMaster](#) (uint32_t baseAddr)
Returns whether the I2C module is in master mode.
- static void [I2C_HAL_ClearArbitrationLost](#) (uint32_t baseAddr)
Clears the arbitration lost flag.

Interrupt

- static void [I2C_HAL_SetIntCmd](#) (uint32_t baseAddr, bool enable)
Enables or disables I2C interrupt requests.
- static bool [I2C_HAL.GetIntCmd](#) (uint32_t baseAddr)
Returns whether the I2C interrupts are enabled.
- static bool [I2C_HAL_IsIntPending](#) (uint32_t baseAddr)
Returns the current I2C interrupt flag.
- static void [I2C_HAL_ClearInt](#) (uint32_t baseAddr)
Clears the I2C interrupt if set.

17.2.4 Enumeration Type Documentation

17.2.4.1 enum i2c_status_t

Enumerator

kStatus_I2C_Success I2C operation has no error.

kStatus_I2C_Initialized Current I2C is already initialized by one task.

kStatus_I2C_Fail I2C operation failed.

I2C HAL driver

kStatus_I2C_Busy The master is already performing a transfer.
kStatus_I2C_Timeout The transfer timed out.
kStatus_I2C_ReceivedNak The slave device sent a NAK in response to a byte.
kStatus_I2C_SlaveTxUnderrun I2C Slave TX Underrun error.
kStatus_I2C_SlaveRxOverrun I2C Slave RX Overrun error.
kStatus_I2C_AribtrationLost I2C Arbitration Lost error.
kStatus_I2C_StopSignalFail I2C STOP signal could not release bus.
kStatus_I2C_Idle I2C Slave Bus is Idle.
kStatus_I2C_NoReceiveInProgress Attempt to abort a receiving when no transfer was in progress.
kStatus_I2C_NoSendInProgress Attempt to abort a sending when no transfer was in progress.

17.2.4.2 enum i2c_status_flag_t

17.2.4.3 enum i2c_direction_t

Enumerator

kI2CReceive Master and slave receive.
kI2CSend Master and slave transmit.

17.2.5 Function Documentation

17.2.5.1 void I2C_HAL_Init(uint32_t baseAddr)

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.2 static void I2C_HAL_Enable(uint32_t baseAddr) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.3 static void I2C_HAL_Disable(uint32_t baseAddr) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.4 static void I2C_HAL_SetGlitchWidth (*uint32_t baseAddr, uint8_t glitchWidth*) [inline], [static]

Controls the width of the glitch, in terms of bus clock cycles, that the filter must absorb. The filter does not allow any glitch whose size is less than or equal to this width setting, to pass.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>glitchWidth</i>	Maximum width in bus clock cycles of the glitches that is filtered. Pass zero to disable the glitch filter.

17.2.5.5 static void I2C_HAL_SetWakeupCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

The I2C module can wake the MCU from low power mode with no peripheral bus running when slave address matching occurs.

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>enable</i>	true - Enables the wakeup function in low power mode. false - Normal operation. No interrupt is generated when address matching in low power mode.

17.2.5.6 void I2C_HAL_SetBaudRate (*uint32_t baseAddr, uint32_t sourceClockInHz, uint32_t kbps, uint32_t * absoluteError_Hz*)

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

I2C HAL driver

<i>sourceClockIn-Hz</i>	I2C source input clock in Hertz
<i>kbps</i>	Requested bus frequency in kilohertz. Common values are either 100 or 400.
<i>absoluteError-Hz</i>	If this parameter is not NULL, it is filled in with the difference in Hertz between the requested bus frequency and the closest frequency possible given available divider values.

17.2.5.7 static void I2C_HAL_SetFreqDiv (uint32_t *baseAddr*, uint8_t *mult*, uint8_t *icr*) [inline], [static]

Use this function to set the I2C bus frequency register values directly, if they are known in advance.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>mult</i>	Value of the MULT bitfield, ranging from 0-2.
<i>icr</i>	The ICR bitfield value, which is the index into an internal table in the I2C hardware that selects the baud rate divisor and SCL hold time.

17.2.5.8 static void I2C_HAL_SetSlaveBaudCtrlCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Enables an independent slave mode baud rate at the maximum frequency. This forces clock stretching on the SCL in very fast I2C modes.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>enable</i>	true - Slave baud rate is independent of the master baud rate; false - The slave baud rate follows the master baud rate and clock stretching may occur.

17.2.5.9 void I2C_HAL_SendStart (uint32_t *baseAddr*)

This function is used to initiate a new master mode transfer by sending the START signal. It is also used to send a Repeated START signal when a transfer is already in progress.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.10 **i2c_status_t I2C_HAL_SendStop (uint32_t *baseAddr*)**

This function changes the direction to receive.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

Returns

Whether the sending of STOP single is success or not.

17.2.5.11 **static void I2C_HAL_SendAck (uint32_t *baseAddr*) [inline], [static]**

This function specifies that an ACK signal is sent in response to the next received byte.

Note that the behavior of this function is changed when the I2C peripheral is placed in Fast ACK mode. In this case, this function causes an ACK signal to be sent in response to the current byte, rather than the next received byte.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.12 **static void I2C_HAL_SendNak (uint32_t *baseAddr*) [inline], [static]**

This function specifies that a NAK signal is sent in response to the next received byte.

Note that the behavior of this function is changed when the I2C peripheral is placed in the Fast ACK mode. In this case, this function causes an NAK signal to be sent in response to the current byte, rather than the next received byte.

Parameters

I2C HAL driver

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.13 static void I2C_HAL_SetDirMode (uint32_t *baseAddr*, i2c_direction_t *direction*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>direction</i>	Specifies either transmit mode or receive mode. The valid values are: <ul style="list-style-type: none">• #kI2CTransmit• kI2CReceive

17.2.5.14 static i2c_direction_t I2C_HAL_GetDirMode (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
-----------------	----------------------------------

Return values

#kI2CTransmit	I2C is configured for master or slave transmit mode.
<i>kI2CReceive</i>	I2C is configured for master or slave receive mode.

17.2.5.15 static uint8_t I2C_HAL_ReadByte (uint32_t *baseAddr*) [inline], [static]

In a master receive mode, calling this function initiates receiving the next byte of data.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

Returns

This function returns the last byte received while the I2C module is configured in master receive or slave receive mode.

17.2.5.16 static void I2C_HAL_WriteByte (uint32_t *baseAddr*, uint8_t *byte*) [inline], [static]

When this function is called in the master transmit mode, a data transfer is initiated. In slave mode, the same function is available after an address match occurs.

In a master transmit mode, the first byte of data written following the start bit or repeated start bit is used for the address transfer and must consist of the slave address (in bits 7-1) concatenated with the required R/#W bit (in position bit 0).

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>byte</i>	The byte of data to transmit.

17.2.5.17 uint8_t I2C_HAL_ReadByteBlocking (uint32_t *baseAddr*)

It will wait till the transfer is actually completed.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

Returns

Returns the last byte received

17.2.5.18 bool I2C_HAL_WriteByteBlocking (uint32_t *baseAddr*, uint8_t *byte*)

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>byte</i>	The byte of data to transmit.

Returns

Whether ACK is received(TRUE) or not(FALSE).

17.2.5.19 i2c_status_t I2C_HAL_MasterReceiveDataPolling (uint32_t *baseAddr*, uint16_t *slaveAddr*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, uint8_t * *rxBuff*, uint32_t *rxSize*)

I2C HAL driver

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>slaveAddr</i>	The slave address to communicate.
<i>cmdBuff</i>	The pointer to the commands to be transferred.
<i>cmdSize</i>	The length in bytes of the commands to be transferred.
<i>rxBuff</i>	The pointer to the data to be transferred.
<i>rxSize</i>	The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

17.2.5.20 i2c_status_t I2C_HAL_MasterSendDataPolling (uint32_t *baseAddr*, uint16_t *slaveAddr*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, const uint8_t * *txBuff*, uint32_t *txSize*)

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>slaveAddr</i>	The slave address to communicate.
<i>cmdBuff</i>	The pointer to the commands to be transferred.
<i>cmdSize</i>	The length in bytes of the commands to be transferred.
<i>txBuff</i>	The pointer to the data to be transferred.
<i>txSize</i>	The length in bytes of the data to be transferred.

Returns

Error or success status returned by API.

17.2.5.21 i2c_status_t I2C_HAL_SlaveSendDataPolling (uint32_t *baseAddr*, const uint8_t * *txBuff*, uint32_t *txSize*)

Parameters

<i>baseAddr</i>	I2C module base address.
<i>txBuff</i>	The buffer pointer which saves the data to be sent.
<i>txSize</i>	Size of data to be sent in unit of byte.

Return values

<i>kStatus_I2C_Received-Nak</i>	if received NACK bit
<i>Error</i>	or success status returned by API.

17.2.5.22 **i2c_status_t I2C_HAL_SlaveReceiveDataPolling (uint32_t *baseAddr*, uint8_t * *rxBuff*, uint32_t *rxSize*)**

Parameters

<i>baseAddr</i>	I2C module base address.
<i>rxBuff</i>	The buffer pointer which saves the data to be received.
<i>rxSize</i>	Size of data need to be received in unit of byte.

Return values

<i>Error</i>	or success status returned by API.
--------------	------------------------------------

17.2.5.23 **void I2C_HAL_SetAddress7bit (uint32_t *baseAddr*, uint8_t *address*)**

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>address</i>	The slave address in the upper 7 bits. Bit 0 of this value must be 0.

17.2.5.24 **void I2C_HAL_SetAddress10bit (uint32_t *baseAddr*, uint16_t *address*)**

I2C HAL driver

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>address</i>	The 10-bit slave address, in bits [10:1] of the value. Bit 0 must be 0.

**17.2.5.25 static void I2C_HAL_SetExtensionAddrCmd (uint32_t *baseAddr*, bool *enable*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>enable</i>	true: 10-bit address is enabled. false: 10-bit address is not enabled.

17.2.5.26 static bool I2C_HAL_GetExtensionAddrCmd (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

Returns

true: 10-bit address is enabled. false: 10-bit address is not enabled.

**17.2.5.27 static void I2C_HAL_SetGeneralCallCmd (uint32_t *baseAddr*, bool *enable*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>enable</i>	Whether to enable the general call address.

**17.2.5.28 static void I2C_HAL_SetRangeMatchCmd (uint32_t *baseAddr*, bool *enable*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>enable</i>	Pass true to enable range address matching. You must also call I2C_HAL_SetUpperAddress7bit() to set the upper address.

17.2.5.29 static void I2C_HAL_SetUpperAddress7bit (uint32_t *baseAddr*, uint8_t *address*) [inline], [static]

This slave address is used as a secondary slave address. If range address matching is enabled, this slave address acts as the upper bound on the slave address range.

This function sets only a 7-bit slave address. If 10-bit addressing was enabled by calling [I2C_HAL_SetAddress10bit\(\)](#), then the top 3 bits set with that function are also used with the address set with this function to form a 10-bit address.

Passing 0 for the *address* parameter disables matching the upper slave address.

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>address</i>	The upper slave address in the upper 7 bits. Bit 0 of this value must be 0. In addition, this address must be greater than the primary slave address that is set by calling I2C_HAL_SetAddress7bit() .

17.2.5.30 static bool I2C_HAL_GetStatusFlag (uint32_t *baseAddr*, i2c_status_flag_t *statusFlag*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
<i>statusFlag</i>	The status flag, defined in type i2c_status_flag_t.

Returns

State of the status flag: asserted (true) or not-asserted (false).

- true: related status flag is being set.
- false: related status flag is not set.

17.2.5.31 static bool I2C_HAL_IsMaster (uint32_t *baseAddr*) [inline], [static]

I2C HAL driver

Parameters

<i>baseAddr</i>	The I2C peripheral base address.
-----------------	----------------------------------

Return values

<i>true</i>	The module is in master mode, which implies it is also performing a transfer.
<i>false</i>	The module is in slave mode.

17.2.5.32 static void I2C_HAL_ClearArbitrationLost (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

17.2.5.33 static void I2C_HAL_SetIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
<i>enable</i>	Pass true to enable interrupt, false to disable.

17.2.5.34 static bool I2C_HAL_GetIntCmd (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

Return values

<i>true</i>	I2C interrupts are enabled.
-------------	-----------------------------

<i>false</i>	I2C interrupts are disabled.
--------------	------------------------------

17.2.5.35 static bool I2C_HAL_IsIntPending (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

Return values

<i>true</i>	An interrupt is pending.
<i>false</i>	No interrupt is pending.

17.2.5.36 static void I2C_HAL_ClearInt (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The I2C peripheral base address
-----------------	---------------------------------

I2C Slave peripheral driver

17.3 I2C Slave peripheral driver

17.3.1 Overview

This section describes the programming interface of the I2C slave mode Peripheral driver.

Data Structures

- struct `i2c_slave_state_t`
Runtime state of the I2C Slave driver. [More...](#)
- struct `i2c_slave_user_config_t`
Defines the structure to initialize the I2C Slave module. [More...](#)

Typedefs

- typedef void(* `i2c_slave_callback_t`) (uint8_t instance, `i2c_slave_event_t` slaveEvent, void *userData)
I2C slave callback function.

Enumerations

- enum `i2c_slave_event_t` {
 `slaveTxReq` = 0x02u,
 `slaveRxReq` = 0x03u,
 `slaveTxNAK` = 0x04u,
 `slaveTxEmpty` = 0x05u,
 `slaveRxFull` = 0x06u }
Internal driver state information.
- enum `i2c_slave_event_flag_t` {
 `kI2CSlaveTxDone` = 0x01u,
 `kI2CSlaveRxDone` = 0x02u }
Defines the type of the enumerating event flag for I2C Slave.

Variables

- const uint32_t `g_i2cBaseAddr` [HW_I2C_INSTANCE_COUNT]
Table of base addresses for I2C instances.

I2C Slave

- void `I2C_DRV_SlaveInit` (uint32_t instance, const `i2c_slave_user_config_t` *userConfigPtr, `i2c_slave_state_t` *slave)
Initializes the I2C module.

- void **I2C_DRV_SlaveDeinit** (uint32_t instance)
Shuts down the I2C slave driver.
- **i2c_slave_state_t * I2C_DRV_SlaveGetHandler** (uint32_t instance)
Get i2c slave run-time state structure.
- **i2c_status_t I2C_DRV_SlaveSendData** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize)
Sends (transmits) data using a non-blocking method.
- **i2c_status_t I2C_DRV_SlaveSendDataBlocking** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize, uint32_t timeout_ms)
Sends (transmits) data using a blocking method.
- **i2c_status_t I2C_DRV_SlaveReceiveData** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize)
Receive the data using a non-blocking method.
- **i2c_status_t I2C_DRV_SlaveReceiveDataBlocking** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize, uint32_t timeout_ms)
Receive the data using a blocking method.
- **i2c_status_t I2C_DRV_SlaveGetReceiveStatus** (uint32_t instance, uint32_t *bytesRemaining)
Gets current status of I2C slave driver .
- **i2c_status_t I2C_DRV_SlaveGetTransmitStatus** (uint32_t instance, uint32_t *bytesRemaining)
Gets current status of I2C slave driver.
- **i2c_status_t I2C_DRV_SlaveAbortReceiveData** (uint32_t instance, uint32_t *rxSize)
Terminates a non-blocking receiving I2C Slave early.
- **i2c_status_t I2C_DRV_SlaveAbortSendData** (uint32_t instance, uint32_t *txSize)
Terminates a non-blocking sending I2C Slave early.
- static bool **I2C_DRV_SlaveIsBusBusy** (uint32_t instance)
Get current status bus of I2C Slave.
- static **i2c_status_t I2C_DRV_SlaveSendDataPolling** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize)
Send out multiple bytes of data using polling method.
- static **i2c_status_t I2C_DRV_SlaveReceiveDataPolling** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize)
Receive multiple bytes of data using polling method.

17.3.2 Data Structure Documentation

17.3.2.1 struct i2c_slave_state_t

This structure holds data used by the I2C Slave Peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress.

Data Fields

- **i2c_status_t status**
The slave I2C status.
- volatile uint32_t **txSize**
Size of the TX buffer.
- volatile uint32_t **rxSize**
Size of the RX buffer.

I2C Slave peripheral driver

- const uint8_t * **txBuff**
Pointer to Tx Buffer.
- uint8_t * **rxBuff**
Pointer to Rx Buffer.
- bool **isTxBusy**
True if the driver is sending data.
- bool **isRxBusy**
True if the driver is receiving data.
- bool **isTxBlocking**
True if transmit is blocking transaction.
- bool **isRxBlocking**
True if receive is blocking transaction.
- event_t **irqEvent**
Use to wait for ISR to complete its Tx, Rx business.
- bool **slaveListening**
True if slave is in listening mode.
- i2c_slave_callback_t **slaveCallback**
Pointer to user callback function.
- void * **callbackParam**
Pointer to user callback param.

17.3.2.1.0.35 Field Documentation

17.3.2.1.0.35.1 i2c_status_t i2c_slave_state_t::status

17.3.2.1.0.35.2 volatile uint32_t i2c_slave_state_t::txSize

17.3.2.1.0.35.3 volatile uint32_t i2c_slave_state_t::rxSize

17.3.2.1.0.35.4 const uint8_t* i2c_slave_state_t::txBuff

17.3.2.1.0.35.5 uint8_t* i2c_slave_state_t::rxBuff

17.3.2.1.0.35.6 bool i2c_slave_state_t::isTxBusy

17.3.2.1.0.35.7 bool i2c_slave_state_t::isRxBusy

17.3.2.1.0.35.8 bool i2c_slave_state_t::isTxBlocking

17.3.2.1.0.35.9 bool i2c_slave_state_t::isRxBlocking

17.3.2.1.0.35.10 bool i2c_slave_state_t::slaveListening

17.3.2.1.0.35.11 i2c_slave_callback_t i2c_slave_state_t::slaveCallback

17.3.2.1.0.35.12 void* i2c_slave_state_t::callbackParam

17.3.2.2 struct i2c_slave_user_config_t

Note

once slaveListening mode is selected, all send/receive blocking/non-blocking functions will be invalid.

Data Fields

- `uint16_t address`
Slave's 7-bit or 10-bit address.
- `bool slaveListening`
The slave configuration mode.
- `i2c_slave_callback_t slaveCallback`
The slave callback function.
- `void * callbackParam`
The slave callback data.

17.3.2.2.0.36 Field Documentation**17.3.2.2.0.36.1 `uint16_t i2c_slave_user_config_t::address`**

If 10-bit address, the first 6 bits must be 011110 in binary.

17.3.2.2.0.36.2 `bool i2c_slave_user_config_t::slaveListening`**17.3.2.2.0.36.3 `i2c_slave_callback_t i2c_slave_user_config_t::slaveCallback`****17.3.2.2.0.36.4 `void* i2c_slave_user_config_t::callbackParam`****17.3.3 Enumeration Type Documentation****17.3.3.1 `enum i2c_slave_event_t`**

Note

The contents of this structure are internal to the driver and should not be modified by users. Also, contents of the structure are subject to change in future releases.

Defines the type of flags for callback function

Enumerator

- `slaveTxReq`** The slave I2C Transmitting Request event.
- `slaveRxReq`** The slave I2C Receiving Request event.
- `slaveTxNAK`** The slave I2C Transmitting NAK event.
- `slaveTxEmpty`** The slave I2C Transmitting Buffer Empty event.
- `slaveRxFull`** The slave I2C Receiving Buffer Full event.

I2C Slave peripheral driver

17.3.3.2 enum i2c_slave_event_flag_t

Enumerator

kI2CSlaveTxDone The slave I2C Transmitting done flag.

kI2CSlaveRxDone The slave I2C Receiving done flag.

17.3.4 Function Documentation

17.3.4.1 void I2C_DRV_SlaveInit (uint32_t *instance*, const i2c_slave_user_config_t * *userConfigPtr*, i2c_slave_state_t * *slave*)

Saves the application callback info, turns on the clock to the module, enables the device, and enables interrupts. Sets the I2C to slave mode. IOMUX should be handled in the [init_hardware\(\)](#) function.

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>userConfigPtr</i>	Pointer of the user configuration structure
<i>slave</i>	Pointer of the slave run-time structure.

17.3.4.2 void I2C_DRV_SlaveDeinit (uint32_t *instance*)

Clears the control register and turns off the clock to the module.

Parameters

<i>instance</i>	Instance number of the I2C module.
-----------------	------------------------------------

17.3.4.3 i2c_slave_state_t* I2C_DRV_SlaveGetHandler (uint32_t *instance*)

This function gets the i2c slave run-time state structure.

Parameters

<i>instance</i>	Instance number of the I2C module.
-----------------	------------------------------------

Returns

Pointer to i2c slave run-time state structure.

17.3.4.4 i2c_status_t I2C_DRV_SlaveSendData (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*)

This function returns immediately when set buffer pointer and length to Tx buffer and Tx Size. The user must check the status of I2C slave to know the whether transmission is finished or not. User can also wait kI2CSlaveStop or kI2CSlaveTxDone to ensure that the transmission is end.

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>txBuff</i>	pointer to sending data buffer.
<i>txSize</i>	the number bytes which user wants to send.

Returns

success (if I2C slave status is not error) or error code in others.

17.3.4.5 i2c_status_t I2C_DRV_SlaveSendDataBlocking (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*, uint32_t *timeout_ms*)

This function set buffer pointer and length to Tx buffer and Tx Size and wait until transmission is end (all data are sent out or STOP signal is detected)

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>txBuff</i>	pointer to sending data buffer.
<i>txSize</i>	the number bytes which user wants to send.
<i>timeout_ms</i>	The maximum number of milliseconds to wait for transmit completed

Returns

success (if I2C slave status is not error) or error code in others.

17.3.4.6 i2c_status_t I2C_DRV_SlaveReceiveData (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*)

This function returns immediately when set buffer pointer and length to Rx buffer and Rx Size. The user must check the status of I2C slave to know the whether transmission is finished or not. User can also wait kI2CSlaveStop or kI2CSlaveRxDone to ensure that the transmission is end.

I2C Slave peripheral driver

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>rxBuff</i>	pointer to received data buffer.
<i>rxSize</i>	the number bytes which user wants to receive.

Returns

success (if I2C slave status is not error) or error code in others.

17.3.4.7 i2c_status_t I2C_DRV_SlaveReceiveDataBlocking (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*, uint32_t *timeout_ms*)

This function set buffer pointer and length to Rx buffer &Rx Size. Then wait until the transmission is end (all data are received or STOP signal is detected)

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>rxBuff</i>	pointer to received data buffer.
<i>rxSize</i>	the number bytes which user wants to receive.
<i>timeout_ms</i>	The maximum number of milliseconds to wait for receive completed

Returns

success (if I2C slave status is not error) or error code in others.

17.3.4.8 i2c_status_t I2C_DRV_SlaveGetReceiveStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>bytes-Remaining</i>	The number of remaining bytes that I2C transmits.

Returns

The current status of I2C instance: in progress (busy), complete (success) or idle (i2c bus is idle).

17.3.4.9 **i2c_status_t I2C_DRV_SlaveGetTransmitStatus (uint32_t *instance*, uint32_t *
bytesRemaining)**

I2C Slave peripheral driver

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>bytes-Remaining</i>	The number of remaining bytes that I2C transmits.

Returns

The current status of I2C instance: in progress (busy), complete (success) or idle (i2c bus is idle).

17.3.4.10 i2c_status_t I2C_DRV_SlaveAbortReceiveData (uint32_t *instance*, uint32_t * *rxSize*)

During an non-blocking receiving

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>rxSize</i>	The number of remaining bytes in I2C Rx Buffer.

Returns

kStatus_I2C_Success if success kStatus_I2C_NoReceiveInProgress if none receiving is available.

17.3.4.11 i2c_status_t I2C_DRV_SlaveAbortSendData (uint32_t *instance*, uint32_t * *txSize*)

During an non-blocking receiving

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>txSize</i>	The number of remaining bytes in I2C Tx Buffer.

Returns

kStatus_I2C_Success if success kStatus_I2C_NoReceiveInProgress if none receiving is available.

17.3.4.12 static bool I2C_DRV_SlavesBusBusy (uint32_t *instance*) [inline], [static]

Parameters

<i>instance</i>	Instance number of the I2C module.
-----------------	------------------------------------

Returns

true if bus is busy false if bus is idle

17.3.4.13 static i2c_status_t I2C_DRV_SlaveSendDataPolling (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*) [inline], [static]

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>txBuff</i>	The buffer pointer which saves the data to be sent.
<i>txSize</i>	Size of data to be sent in unit of byte.

Returns

Error or success status returned by API.

17.3.4.14 static i2c_status_t I2C_DRV_SlaveReceiveDataPolling (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*) [inline], [static]

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>rxBuff</i>	The buffer pointer which saves the data to be received.
<i>rxSize</i>	Size of data need to be received in unit of byte.

Returns

Error or success status returned by API.

17.3.5 Variable Documentation

17.3.5.1 const uint32_t g_i2cBaseAddr[HW_I2C_INSTANCE_COUNT]

I2C Master peripheral

17.4 I2C Master peripheral

17.4.1 Overview

This section describes the programming interface of the I2C master mode Peripheral driver. The I2C master Peripheral driver provides functions for a master device to send and receive data.

17.4.2 I2C Initialization

To initialize the I2C driver, define an `i2c_master_state_t` type variable and pass it in the `i2c_init`. The variable does not need to have a specific value because the I2C driver only needs the memory associated with the variable. Because all I2C master API functions need the run-time structure, it should be maintained as long as the driver is used.

Because I2C drivers use the OSA_Delay form OSA layer, `OSA_Init` must be called before calling I2C transaction API functions. Otherwise, the API functions do not work.

17.4.3 I2C Data Transactions

I2C master driver mainly provides two APIs for data transactions:

```
I2C_DRV_MasterSendDataBlocking \n
I2C_DRV_MasterReceiveDataBlocking \n
```

Both of these functions perform a blocking transaction, which means that the function does not return until all data is sent/received OR a time out occurs.

Before calling the API functions, the application should define the slave device with "`i2c_device_t`" type. Note that, if the slave is a 10-bit address, the first 6 bits must be 011110 in binary.

```
typedef struct I2CDevice
{
    uint16_t address; // Slave's 7-bit or 10-bit address. If 10-bit address, the first 6 bits must be
                      // 011110 in binary.//
    uint32_t baudRate_kbps; // The baud rate in kbps to use with this slave device.//
} i2c_device_t;
```

If the `baudRate_kbps` inside the `i2c_device_t` object is changed, it is automatically applied to the next send/receive transaction.

Data Structures

- struct `i2c_device_t`
Information necessary to communicate with an I2C slave device. [More...](#)
- struct `i2c_master_state_t`
Internal driver state information. [More...](#)

Variables

- const uint32_t [g_i2cBaseAddr](#) [HW_I2C_INSTANCE_COUNT]
Table of base addresses for I2C instances.

I2C Master

- [i2c_status_t I2C_DRV_MasterInit](#) (uint32_t instance, [i2c_master_state_t](#) *master)
Initializes the I2C master mode driver.
- void [I2C_DRV_MasterDeinit](#) (uint32_t instance)
Shuts down the driver.
- void [I2C_DRV_MasterSetBaudRate](#) (uint32_t instance, const [i2c_device_t](#) *device)
Configures the I2C bus to access a device.
- [i2c_status_t I2C_DRV_MasterSendDataBlocking](#) (uint32_t instance, const [i2c_device_t](#) *device, const uint8_t *cmdBuff, uint32_t cmdSize, const uint8_t *txBuff, uint32_t txSize, uint32_t timeout_ms)
Performs a blocking send transaction on the I2C bus.
- [i2c_status_t I2C_DRV_MasterSendData](#) (uint32_t instance, const [i2c_device_t](#) *device, const uint8_t *cmdBuff, uint32_t cmdSize, const uint8_t *txBuff, uint32_t txSize)
Performs a non-blocking send transaction on the I2C bus.
- [i2c_status_t I2C_DRV_MasterGetSendStatus](#) (uint32_t instance, uint32_t *bytesRemaining)
Gets current status of I2C master transmit.
- [i2c_status_t I2C_DRV_MasterAbortSendData](#) (uint32_t instance)
Terminates a non-blocking I2C Master transmission early.
- [i2c_status_t I2C_DRV_MasterReceiveDataBlocking](#) (uint32_t instance, const [i2c_device_t](#) *device, const uint8_t *cmdBuff, uint32_t cmdSize, uint8_t *rxBuff, uint32_t rxSize, uint32_t timeout_ms)
Performs a blocking receive transaction on the I2C bus.
- [i2c_status_t I2C_DRV_MasterReceiveData](#) (uint32_t instance, const [i2c_device_t](#) *device, const uint8_t *cmdBuff, uint32_t cmdSize, uint8_t *rxBuff, uint32_t rxSize)
Performs a non-blocking receive transaction on the I2C bus.
- [i2c_status_t I2C_DRV_MasterGetReceiveStatus](#) (uint32_t instance, uint32_t *bytesRemaining)
Gets current status of I2C master receiving.
- static [i2c_status_t I2C_DRV_MasterReceiveDataPolling](#) (uint32_t instance, uint16_t slaveAddr, const uint8_t *cmdBuff, uint32_t cmdSize, uint8_t *rxBuff, uint32_t rxSize)
Performs a polling receive transaction on the I2C bus.
- static [i2c_status_t I2C_DRV_MasterSendDataPolling](#) (uint32_t instance, uint16_t slaveAddr, const uint8_t *cmdBuff, uint32_t cmdSize, const uint8_t *txBuff, uint32_t txSize)
Performs a polling send transaction on the I2C bus.

17.4.4 Data Structure Documentation

17.4.4.1 struct i2c_device_t

Data Fields

- uint16_t address

I2C Master peripheral

- Slave's 7-bit or 10-bit address.
- uint32_t **baudRate_kbps**
The baud rate in kbps to use by current slave device.

17.4.4.1.0.37 Field Documentation

17.4.4.1.0.37.1 uint16_t i2c_device_t::address

If 10-bit address, the first 6 bits must be 011110 in binary.

17.4.4.1.0.37.2 uint32_t i2c_device_t::baudRate_kbps

17.4.4.2 struct i2c_master_state_t

Note

The contents of this structure are internal to the driver and should not be modified by users. Also, contents of the structure are subject to change in future releases.

17.4.5 Function Documentation

17.4.5.1 i2c_status_t I2C_DRV_MasterInit (uint32_t *instance*, i2c_master_state_t * *master*)

Parameters

<i>instance</i>	The I2C peripheral instance number.
<i>master</i>	The pointer to the I2C master driver state structure.

Returns

Error or success status returned by API.

17.4.5.2 void I2C_DRV_MasterDeinit (uint32_t *instance*)

Parameters

<i>instance</i>	The I2C peripheral instance number.
-----------------	-------------------------------------

17.4.5.3 void I2C_DRV_MasterSetBaudRate (uint32_t *instance*, const i2c_device_t * *device*)

Parameters

<i>instance</i>	The I2C peripheral instance number.
<i>device</i>	The pointer to the I2C device information structure.

17.4.5.4 **i2c_status_t I2C_DRV_MasterSendDataBlocking (uint32_t *instance*, const i2c_device_t * *device*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, const uint8_t * *txBuff*, uint32_t *txSize*, uint32_t *timeout_ms*)**

Both cmdBuff and txBuff are byte aligned, user needs to prepare these buffers according to related protocol if slave devices data are not byte-aligned.

Parameters

<i>instance</i>	The I2C peripheral instance number.
<i>device</i>	The pointer to the I2C device information structure.
<i>cmdBuff</i>	The pointer to the commands to be transferred, could be NULL.
<i>cmdSize</i>	The length in bytes of the commands to be transferred, could be 0.
<i>txBuff</i>	The pointer to the data to be transferred, cannot be NULL.
<i>txSize</i>	The length in bytes of the data to be transferred, cannot be 0.
<i>timeout_ms</i>	A timeout for the transfer in microseconds.

Returns

Error or success status returned by API.

17.4.5.5 **i2c_status_t I2C_DRV_MasterSendData (uint32_t *instance*, const i2c_device_t * *device*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, const uint8_t * *txBuff*, uint32_t *txSize*)**

This function returns immediately when set buffer pointer and length to Tx buffer and Tx Size. The user must check the status of I2C to know the whether transmission is finished or not. Both cmdBuff and txBuff are byte aligned, user needs to prepare these buffers according to related protocol if slave devices data are not byte-aligned.

Parameters

I2C Master peripheral

<i>instance</i>	The I2C peripheral instance number.
<i>device</i>	The pointer to the I2C device information structure.
<i>cmdBuff</i>	The pointer to the commands to be transferred, could be NULL.
<i>cmdSize</i>	The length in bytes of the commands to be transferred, could be 0.
<i>txBuff</i>	The pointer to the data to be transferred, cannot be NULL.
<i>txSize</i>	The length in bytes of the data to be transferred, cannot be 0.

Returns

Error or success status returned by API.

17.4.5.6 i2c_status_t I2C_DRV_MasterGetSendStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)

This function is designed to get current I2C status of non-blocking transmit.

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>bytes-Remaining</i>	The number of remaining bytes in the active I2C transmits.

Returns

Current status of I2C transmission: in progress (busy) or complete (success).

17.4.5.7 i2c_status_t I2C_DRV_MasterAbortSendData (uint32_t *instance*)

Parameters

<i>instance</i>	Instance number of the I2C module.
-----------------	------------------------------------

Returns

Whether the aborting is success or not.

17.4.5.8 **i2c_status_t I2C_DRV_MasterReceiveDataBlocking (uint32_t *instance*, const i2c_device_t * *device*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, uint8_t * *rxBuff*, uint32_t *rxSize*, uint32_t *timeout_ms*)**

Both cmdBuff and rxBuff are byte aligned, user needs to prepare these buffers according to related protocol if slave devices data are not byte-aligned.

I2C Master peripheral

Parameters

<i>instance</i>	The I2C peripheral instance number.
<i>device</i>	The pointer to the I2C device information structure.
<i>cmdBuff</i>	The pointer to the commands to be transferred, could be NULL.
<i>cmdSize</i>	The length in bytes of the commands to be transferred, could be 0.
<i>rxBuff</i>	The pointer to the data to be transferred, cannot be NULL.
<i>rxSize</i>	The length in bytes of the data to be transferred, cannot be 0.
<i>timeout_ms</i>	A timeout for the transfer in microseconds.

Returns

Error or success status returned by API.

17.4.5.9 i2c_status_t I2C_DRV_MasterReceiveData (uint32_t *instance*, const i2c_device_t * *device*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, uint8_t * *rxBuff*, uint32_t *rxSize*)

This function returns immediately after set buffer pointer and length to Rx buffer and Rx Size. The user must check the status of I2C to know the whether the receiving is finished or not. Both cmdBuff and rxBuff are byte aligned, user needs to prepare these buffers according to related protocol if slave devices data are not byte-aligned.

Parameters

<i>instance</i>	The I2C peripheral instance number.
<i>device</i>	The pointer to the I2C device information structure.
<i>cmdBuff</i>	The pointer to the commands to be transferred, could be NULL.
<i>cmdSize</i>	The length in bytes of the commands to be transferred, could be 0.
<i>rxBuff</i>	The pointer to the data to be transferred, cannot be NULL.
<i>rxSize</i>	The length in bytes of the data to be transferred, cannot be 0.

Returns

Error or success status returned by API.

17.4.5.10 i2c_status_t I2C_DRV_MasterGetReceiveStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)

This function is designed to get current I2C status of non-blocking receive.

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>bytes-Remaining</i>	The number of remaining bytes in the active I2C transmits.

Returns

Current status of I2C receive: in progress (busy) or complete (success).

17.4.5.11 static i2c_status_t I2C_DRV_MasterReceiveDataPolling (uint32_t *instance*, uint16_t *slaveAddr*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, uint8_t * *rxBuff*, uint32_t *rxSize*) [inline], [static]

Both cmdBuff and rxBuff are byte aligned, user needs to prepare these buffers according to related protocol if slave devices data are not byte-aligned.

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>slaveAddr</i>	The slave address to communicate.
<i>cmdBuff</i>	The pointer to the commands to be transferred, could be NULL.
<i>cmdSize</i>	The length in bytes of the commands to be transferred, could be 0.
<i>rxBuff</i>	The pointer to the data to be transferred, cannot be NULL.
<i>rxSize</i>	The length in bytes of the data to be transferred, cannot be 0.

Returns

Error or success status returned by API.

17.4.5.12 static i2c_status_t I2C_DRV_MasterSendDataPolling (uint32_t *instance*, uint16_t *slaveAddr*, const uint8_t * *cmdBuff*, uint32_t *cmdSize*, const uint8_t * *txBuff*, uint32_t *txSize*) [inline], [static]

Both cmdBuff and txBuff are byte aligned, user needs to prepare these buffers according to related protocol if slave devices data are not byte-aligned.

I2C Master peripheral

Parameters

<i>instance</i>	Instance number of the I2C module.
<i>slaveAddr</i>	The slave address to communicate.
<i>cmdBuff</i>	The pointer to the commands to be transferred, could be NULL.
<i>cmdSize</i>	The length in bytes of the commands to be transferred, could be 0.
<i>txBuff</i>	The pointer to the data to be transferred, cannot be NULL.
<i>txSize</i>	The length in bytes of the data to be transferred, cannot be 0.

Returns

Error or success status returned by API.

17.4.6 Variable Documentation

17.4.6.1 `const uint32_t g_i2cBaseAddr[HW_I2C_INSTANCE_COUNT]`

Chapter 18

Universal Asynchronous Receiver/Transmitter (UART)

18.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Universal Asynchronous Receiver/-Transmitter (UART) block of Kinetis devices.

Modules

- [UART HAL driver](#)
- [UART Peripheral driver](#)

18.2 UART HAL driver

18.2.1 Overview

The section describes the programming interface of the UART HAL driver.

Files

- file [fsl_uart_hal.h](#)

Enumerations

- enum [uart_status_t](#)
Error codes for the UART driver.
- enum [uart_stop_bit_count_t](#) {
 kUartOneStopBit = 0U,
 kUartTwoStopBit = 1U }
UART number of stop bits.
- enum [uart_parity_mode_t](#) {
 kUartParityDisabled = 0x0U,
 kUartParityEven = 0x2U,
 kUartParityOdd = 0x3U }
UART parity mode.
- enum [uart_bit_count_per_char_t](#) {
 kUart8BitsPerChar = 0U,
 kUart9BitsPerChar = 1U }
UART number of bits in a character.
- enum [uart_operation_config_t](#) {
 kUartOperates = 0U,
 kUartStops = 1U }
UART operation configuration constants.
- enum [uart_receiver_source_t](#) {
 kUartLoopBack = 0U,
 kUartSingleWire = 1U }
UART receiver source select mode.
- enum [uart_wakeup_method_t](#) {
 kUartIdleLineWake = 0U,
 kUartAddrMarkWake = 1U }
UART wakeup from standby method constants.
- enum [uart_idle_line_select_t](#) {
 kUartIdleLineAfterStartBit = 0U,
 kUartIdleLineAfterStopBit = 1U }
UART idle-line detect selection types.
- enum [uart_break_char_length_t](#) {
 kUartBreakChar10BitMinimum = 0U,
 kUartBreakChar13BitMinimum = 1U }

- *UART break character length settings for transmit/detect.*
- enum `uart_singlewire_txdir_t` {

 `kUartSinglewireTxdirIn` = 0U,

 `kUartSinglewireTxdirOut` = 1U }
- UART single-wire mode transmit direction.*
- enum `uart_ir_tx_pulsewidth_t` {

 `kUartIrThreeSixteenthsWidth` = 0U,

 `kUartIrOneSixteenthWidth` = 1U,

 `kUartIrOneThirtysecondsWidth` = 2U,

 `kUartIrOneFourthWidth` = 3U }
- UART infrared transmitter pulse width options.*
- enum `uart_status_flag_t` {

 `kUartTxDataRegEmpty` = 0U << UART_SHIFT | BP_UART_S1_TDRE,

 `kUartTxComplete` = 0U << UART_SHIFT | BP_UART_S1_TC,

 `kUartRxDataRegFull` = 0U << UART_SHIFT | BP_UART_S1_RDRF,

 `kUartIdleLineDetect` = 0U << UART_SHIFT | BP_UART_S1_IDLE,

 `kUartRxOverrun` = 0U << UART_SHIFT | BP_UART_S1_OR,

 `kUartNoiseDetect` = 0U << UART_SHIFT | BP_UART_S1_NF,

 `kUartFrameErr` = 0U << UART_SHIFT | BP_UART_S1_FE,

 `kUartParityErr` = 0U << UART_SHIFT | BP_UART_S1_PF,

 `kUartRxActiveEdgeDetect` = 1U << UART_SHIFT | BP_UART_S2_RXEDGIF,

 `kUartRxActive` = 1U << UART_SHIFT | BP_UART_S2_RAF }
- UART status flags.*
- enum `uart_interrupt_t` {

 `kUartIntRxActiveEdge` = 0U << UART_SHIFT | BP_UART_BDH_RXEDGIE,

 `kUartIntTxDataRegEmpty` = 1U << UART_SHIFT | BP_UART_C2_TIE,

 `kUartIntTxComplete` = 1U << UART_SHIFT | BP_UART_C2_TCIE,

 `kUartIntRxDataRegFull` = 1U << UART_SHIFT | BP_UART_C2_RIE,

 `kUartIntIdleLine` = 1U << UART_SHIFT | BP_UART_C2_ILIE,

 `kUartIntRxOverrun` = 2U << UART_SHIFT | BP_UART_C3_ORIE,

 `kUartIntNoiseErrFlag` = 2U << UART_SHIFT | BP_UART_C3_NEIE,

 `kUartIntFrameErrFlag` = 2U << UART_SHIFT | BP_UART_C3_FEIE,

 `kUartIntParityErrFlag` = 2U << UART_SHIFT | BP_UART_C3_PEIE }
- UART interrupt configuration structure, default settings are 0 (disabled).*

UART Common Configurations

- void `UART_HAL_Init` (uint32_t baseAddr)

 Initializes the UART controller.
- static void `UART_HAL_EnableTransmitter` (uint32_t baseAddr)

 Enables the UART transmitter.
- static void `UART_HAL_DisableTransmitter` (uint32_t baseAddr)

 Disables the UART transmitter.
- static bool `UART_HAL_IsTransmitterEnabled` (uint32_t baseAddr)

 Gets the UART transmitter enabled/disabled configuration setting.
- static void `UART_HAL_EnableReceiver` (uint32_t baseAddr)

UART HAL driver

- static void **UART_HAL_DisableReceiver** (uint32_t baseAddr)
Disables the UART receiver.
- static bool **UART_HAL_IsReceiverEnabled** (uint32_t baseAddr)
Gets the UART receiver enabled/disabled configuration setting.
- **uart_status_t UART_HAL_SetBaudRate** (uint32_t baseAddr, uint32_t sourceClockInHz, uint32_t baudRate)
Configures the UART baud rate.
- void **UART_HAL_SetBaudRateDivisor** (uint32_t baseAddr, uint16_t baudRateDivisor)
Sets the UART baud rate modulo divisor value.
- static void **UART_HAL_SetBitCountPerChar** (uint32_t baseAddr, **uart_bit_count_per_char_t** bitCountPerChar)
Configures the number of bits per character in the UART controller.
- void **UART_HAL_SetParityMode** (uint32_t baseAddr, **uart_parity_mode_t** parityMode)
Configures the parity mode in the UART controller.
- static void **UART_HAL_SetTxInversionCmd** (uint32_t baseAddr, bool enable)
Enable or disable the transmit inversion control in UART controller.
- static void **UART_HAL_SetRxInversionCmd** (uint32_t baseAddr, bool enable)
Enable or disable the receive inversion control in UART controller.
- static uint32_t **UART_HAL_GetDataRegAddr** (uint32_t baseAddr)
Get UART tx/rx data register address.

UART Interrupts and DMA

- void **UART_HAL_SetIntMode** (uint32_t baseAddr, **uart_interrupt_t** interrupt, bool enable)
Configures the UART module interrupts to enable/disable various interrupt sources.
- bool **UART_HAL.GetIntMode** (uint32_t baseAddr, **uart_interrupt_t** interrupt)
Returns whether the UART module interrupts is enabled/disabled.
- static void **UART_HAL_SetTxDataRegEmptyIntCmd** (uint32_t baseAddr, bool enable)
Enables or disables the tx_data_register_empty_interrupt.
- static bool **UART_HAL.GetTxDataRegEmptyIntCmd** (uint32_t baseAddr)
Gets the configuration of the tx_data_register_empty_interrupt enable setting.
- static void **UART_HAL_SetRxDataRegFullIntCmd** (uint32_t baseAddr, bool enable)
Disables the rx_data_register_full_interrupt.
- static bool **UART_HAL.GetRxDataRegFullIntCmd** (uint32_t baseAddr)
Gets the configuration of the rx_data_register_full_interrupt enable setting.

UART Transfer Functions

- void **UART_HAL_Putchar** (uint32_t baseAddr, uint8_t data)
This function allows the user to send an 8-bit character from the UART data register.
- void **UART_HAL_Putchar9** (uint32_t baseAddr, uint16_t data)
This function allows the user to send a 9-bit character from the UART data register.
- void **UART_HAL_Getchar** (uint32_t baseAddr, uint8_t *readData)
This function gets a received 8-bit character from the UART data register.
- void **UART_HAL_Getchar9** (uint32_t baseAddr, uint16_t *readData)
This function gets a received 9-bit character from the UART data register.
- void **UART_HAL_SendDataPolling** (uint32_t baseAddr, const uint8_t *txBuff, uint32_t txSize)

- *Send out multiple bytes of data using polling method.*
uart_status_t UART_HAL_ReceiveDataPolling (uint32_t baseAddr, uint8_t *rxBuff, uint32_t rxSize)
Receive multiple bytes of data using polling method.

UART Special Feature Configurations

- static void **UART_HAL_SetLoopCmd** (uint32_t baseAddr, bool enable)
Configures the UART loopback operation.
- static void **UART_HAL_SetReceiverSource** (uint32_t baseAddr, **uart_receiver_source_t** source)
Configures the UART single-wire operation.
- static void **UART_HAL_SetTransmitterDir** (uint32_t baseAddr, **uart_singlewire_txdir_t** direction)
Configures the UART transmit direction while in single-wire mode.
- **uart_status_t UART_HAL_PutReceiverInStandbyMode** (uint32_t baseAddr)
Places the UART receiver in standby mode.
- static void **UART_HAL_PutReceiverInNormalMode** (uint32_t baseAddr)
Places the UART receiver in normal mode (disable standby mode operation).
- static bool **UART_HAL_IsReceiverInStandby** (uint32_t baseAddr)
Determines if the UART receiver is currently in standby mode.
- static void **UART_HAL_SetReceiverWakeupMethod** (uint32_t baseAddr, **uart_wakeup_method_t** method)
Selects the UART receiver wakeup method (idle-line or address-mark) from standby mode.
- static **uart_wakeup_method_t UART_HAL_GetReceiverWakeupMethod** (uint32_t baseAddr)
Gets the UART receiver wakeup method (idle-line or address-mark) from standby mode.
- void **UART_HAL_ConfigIdleLineDetect** (uint32_t baseAddr, uint8_t idleLine, uint8_t rxWakeIdleDetect)
Configures the operation options of the UART idle line detect.
- static void **UART_HAL_SetBreakCharTransmitLength** (uint32_t baseAddr, **uart_break_char_length_t** length)
Configures the UART break character transmit length.
- static void **UART_HAL_SetBreakCharCmd** (uint32_t baseAddr, bool enable)
Configures the UART transmit send break character operation.
- void **UART_HAL_SetMatchAddress** (uint32_t baseAddr, bool matchAddrMode1, bool matchAddrMode2, uint8_t matchAddrValue1, uint8_t matchAddrValue2)
Configures the UART match address mode control operation.

UART Status Flags

- bool **UART_HAL_GetStatusFlag** (uint32_t baseAddr, **uart_status_flag_t** statusFlag)
Gets all UART status flag states.
- static bool **UART_HAL_IsTxDataRegEmpty** (uint32_t baseAddr)
Gets the UART Transmit data register empty flag.
- static bool **UART_HAL_IsTxComplete** (uint32_t baseAddr)
Gets the UART Transmission complete flag.
- static bool **UART_HAL_IsRxDataRegFull** (uint32_t baseAddr)
Gets the UART Receive data register full flag.
- **uart_status_t UART_HAL_ClearStatusFlag** (uint32_t baseAddr, **uart_status_flag_t** statusFlag)
Clears an individual and specific UART status flag.

18.2.2 Enumeration Type Documentation

18.2.2.1 enum uart_status_t

18.2.2.2 enum uart_stop_bit_count_t

These constants define the number of allowable stop bits to configure in a UART baseAddr.

Enumerator

kUartOneStopBit one stop bit

kUartTwoStopBit two stop bits

18.2.2.3 enum uart_parity_mode_t

These constants define the UART parity mode options: disabled or enabled of type even or odd.

Enumerator

kUartParityDisabled parity disabled

kUartParityEven parity enabled, type even, bit setting: PE|PT = 10

kUartParityOdd parity enabled, type odd, bit setting: PE|PT = 11

18.2.2.4 enum uart_bit_count_per_char_t

These constants define the number of allowable data bits per UART character. Note, check the UART documentation to determine if the desired UART baseAddr supports the desired number of data bits per UART character.

Enumerator

kUart8BitsPerChar 8-bit data characters

kUart9BitsPerChar 9-bit data characters

18.2.2.5 enum uart_operation_config_t

This provides constants for UART operational states: "operates normally" or "stops/ceases operation"

Enumerator

kUartOperates UART continues to operate normally.

kUartStops UART ceases operation.

18.2.2.6 enum uart_receiver_source_t

Enumerator

kUartLoopBack Internal loop back mode.

kUartSingleWire Single wire mode.

18.2.2.7 enum uart_wakeup_method_t

This provides constants for the two UART wakeup methods: idle-line or address-mark.

Enumerator

kUartIdleLineWake The idle-line wakes UART receiver from standby.

kUartAddrMarkWake The address-mark wakes UART receiver from standby.

18.2.2.8 enum uart_idle_line_select_t

This provides constants for the UART idle character bit-count start: either after start or stop bit.

Enumerator

kUartIdleLineAfterStartBit UART idle character bit count start after start bit.

kUartIdleLineAfterStopBit UART idle character bit count start after stop bit.

18.2.2.9 enum uart_break_char_length_t

This provides constants for the UART break character length for both transmission and detection purposes. Note that the actual maximum bit times may vary depending on the UART baseAddr.

Enumerator

kUartBreakChar10BitMinimum UART break char length 10 bit times (if M = 0, SBNS = 0) or 11 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 12 (if M = 1, SBNS = 1 or M10 = 1, SNBS = 0) or 13 (if M10 = 1, SNBS = 1)

kUartBreakChar13BitMinimum UART break char length 13 bit times (if M = 0, SBNS = 0) or 14 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 15 (if M = 1, SBNS = 1 or M10 = 1, SNBS = 0) or 16 (if M10 = 1, SNBS = 1)

18.2.2.10 enum uart_singlewire_txdir_t

This provides constants for the UART transmit direction when configured for single-wire mode. The transmit line TXDIR is either an input or output.

UART HAL driver

Enumerator

- kUartSinglewireTxdirIn*** UART Single-Wire mode TXDIR input.
- kUartSinglewireTxdirOut*** UART Single-Wire mode TXDIR output.

18.2.2.11 enum uart_ir_tx_pulsewidth_t

This provides constants for the UART infrared (IR) pulse widths. Options include 3/16, 1/16 1/32, and 1/4 pulse widths.

Enumerator

- kUartIrThreeSixteenthsWidth*** 3/16 pulse
- kUartIrOneSixteenthWidth*** 1/16 pulse
- kUartIrOneThirtysecondsWidth*** 1/32 pulse
- kUartIrOneFourthWidth*** 1/4 pulse

18.2.2.12 enum uart_status_flag_t

This provides constants for the UART status flags for use in the UART functions.

Enumerator

- kUartTxDataRegEmpty*** Tx data register empty flag, sets when Tx buffer is empty.
- kUartTxComplete*** Transmission complete flag, sets when transmission activity complete.
- kUartRxDataRegFull*** Rx data register full flag, sets when the receive data buffer is full.
- kUartIdleLineDetect*** Idle line detect flag, sets when idle line detected.
- kUartRxOverrun*** Rrx Overrun, sets when new data is received before data is read from receive register.
- kUartNoiseDetect*** Rrx takes 3 samples of each received bit. If any of these samples differ, noise flag sets
- kUartFrameErr*** Frame error flag, sets if logic 0 was detected where stop bit expected.
- kUartParityErr*** If parity enabled, sets upon parity error detection.
- kUartRxActiveEdgeDetect*** Rx pin active edge interrupt flag, sets when active edge detected.
- kUartRxActive*** Receiver Active Flag (RAF), sets at beginning of valid start bit.

18.2.2.13 enum uart_interrupt_t

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

Enumerator

- kUartIntRxActiveEdge*** RX Active Edge.
- kUartIntTxDataRegEmpty*** Transmit data register empty.

kUartIntTxComplete Transmission complete.
kUartIntRxDataRegFull Receiver data register full.
kUartIntIdleLine Idle line.
kUartIntRxOverrun Receiver Overrun.
kUartIntNoiseErrFlag Noise error flag.
kUartIntFrameErrFlag Framing error flag.
kUartIntParityErrFlag Parity error flag.

18.2.3 Function Documentation

18.2.3.1 void UART_HAL_Init(uint32_t baseAddr)

This function initializes the module to a known state.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

18.2.3.2 static void UART_HAL_EnableTransmitter(uint32_t baseAddr) [inline], [static]

This function allows the user to enable the UART transmitter.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

18.2.3.3 static void UART_HAL_DisableTransmitter(uint32_t baseAddr) [inline], [static]

This function allows the user to disable the UART transmitter.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

18.2.3.4 static bool UART_HAL_IsTransmitterEnabled(uint32_t baseAddr) [inline], [static]

This function allows the user to get the setting of the UART transmitter.

UART HAL driver

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The state of UART transmitter enable(true)/disable(false) setting.

18.2.3.5 static void UART_HAL_EnableReceiver (uint32_t *baseAddr*) [inline], [static]

This function allows the user to enable the UART receiver.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

18.2.3.6 static void UART_HAL_DisableReceiver (uint32_t *baseAddr*) [inline], [static]

This function allows the user to disable the UART receiver.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

18.2.3.7 static bool UART_HAL_IsReceiverEnabled (uint32_t *baseAddr*) [inline], [static]

This function allows the user to get the setting of the UART receiver.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The state of UART receiver enable(true)/disable(false) setting.

18.2.3.8 **uart_status_t UART_HAL_SetBaudRate (uint32_t *baseAddr*, uint32_t *sourceClockInHz*, uint32_t *baudRate*)**

This function programs the UART baud rate to the desired value passed in by the user. The user must also pass in the module source clock so that the function can calculate the baud rate divisors to their appropriate values. In some UART baseAddrs it is required that the transmitter/receiver be disabled before calling this function. Generally this is applied to all UARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	UART module base address.
<i>sourceClockInHz</i>	UART source input clock in Hz.
<i>baudRate</i>	UART desired baud rate.

Returns

An error code or kStatus_UART_Success

18.2.3.9 **void UART_HAL_SetBaudRateDivisor (uint32_t *baseAddr*, uint16_t *baudRateDivisor*)**

This function allows the user to program the baud rate divisor directly in situations where the divisor value is known. In this case, the user may not want to call the [UART_HAL_SetBaudRate\(\)](#) function, as the divisor is already known.

Parameters

<i>baseAddr</i>	UART module base address.
<i>baudRateDivisor</i>	The baud rate modulo division "SBR" value.

18.2.3.10 **static void UART_HAL_SetBitCountPerChar (uint32_t *baseAddr*, uart_bit_count_per_char_t *bitCountPerChar*) [inline], [static]**

This function allows the user to configure the number of bits per character according to the typedef `uart_bit_count_per_char_t`.

UART HAL driver

Parameters

<i>baseAddr</i>	UART module base address.
<i>bitCountPerChar</i>	Number of bits per char (8, 9, or 10, depending on the UART baseAddr).

18.2.3.11 void **UART_HAL_SetParityMode** (**uint32_t baseAddr, uart_parity_mode_t parityMode**)

This function allows the user to configure the parity mode of the UART controller to disable it or enable it for even parity or for odd parity.

Parameters

<i>baseAddr</i>	UART module base address.
<i>parityMode</i>	Parity mode setting (enabled, disable, odd, even - see parity_mode_t struct).

18.2.3.12 static void **UART_HAL_SetTxInversionCmd** (**uint32_t baseAddr, bool enable**) [**inline**], [**static**]

This function allows the user to invert the transmit signals.

Parameters

<i>baseAddr</i>	UART module base address.
<i>enable</i>	Enable (true) or disable (false) transmit inversion.

18.2.3.13 static void **UART_HAL_SetRxInversionCmd** (**uint32_t baseAddr, bool enable**) [**inline**], [**static**]

This function allows the user to invert the receive signals.

Parameters

<i>baseAddr</i>	UART module base address.
<i>enable</i>	Enable (true) or disable (false) receive inversion.

18.2.3.14 static uint32_t **UART_HAL_GetDataRegAddr** (**uint32_t baseAddr**) [**inline**], [**static**]

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

UART tx/rx data register address.

18.2.3.15 void UART_HAL_SetIntMode (*uint32_t baseAddr, uart_interrupt_t interrupt, bool enable*)

Parameters

<i>baseAddr</i>	UART module base address.
<i>interrupt</i>	UART interrupt configuration data.
<i>enable</i>	true: enable, false: disable.

18.2.3.16 bool UART_HAL_GetIntMode (*uint32_t baseAddr, uart_interrupt_t interrupt*)

Parameters

<i>baseAddr</i>	UART module base address.
<i>interrupt</i>	UART interrupt configuration data.

Returns

true: enable, false: disable.

18.2.3.17 static void UART_HAL_SetTxDataRegEmptyIntCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

UART HAL driver

<i>enable</i>	true: enable, false: disable.
---------------	-------------------------------

**18.2.3.18 static bool UART_HAL_GetTxDataRegEmptyIntCmd (uint32_t *baseAddr*)
[inline], [static]**

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

setting of the interrupt enable bit.

**18.2.3.19 static void UART_HAL_SetRxDataRegFullIntCmd (uint32_t *baseAddr*, bool
enable) [inline], [static]**

Parameters

<i>baseAddr</i>	UART module base address.
<i>enable</i>	true: enable, false: disable.

**18.2.3.20 static bool UART_HAL_GetRxDataRegFullIntCmd (uint32_t *baseAddr*)
[inline], [static]**

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

Bit setting of the interrupt enable bit.

18.2.3.21 void UART_HAL_Putchar (uint32_t *baseAddr*, uint8_t *data*)

Parameters

<i>baseAddr</i>	UART module base address.
<i>data</i>	The data to send of size 8-bit.

18.2.3.22 void UART_HAL_Putchar9 (*uint32_t baseAddr, uint16_t data*)

Parameters

<i>baseAddr</i>	UART module base address.
<i>data</i>	The data to send of size 9-bit.

18.2.3.23 void UART_HAL_Getchar (*uint32_t baseAddr, uint8_t * readData*)

Parameters

<i>baseAddr</i>	UART module base address.
<i>readData</i>	The received data read from data register of size 8-bit.

18.2.3.24 void UART_HAL_Getchar9 (*uint32_t baseAddr, uint16_t * readData*)

Parameters

<i>baseAddr</i>	UART module base address.
<i>readData</i>	The received data read from data register of size 9-bit.

18.2.3.25 void UART_HAL_SendDataPolling (*uint32_t baseAddr, const uint8_t * txBuff, uint32_t txSize*)

This function only supports 8-bit transaction.

Parameters

<i>baseAddr</i>	UART module base address.
<i>txBuff</i>	The buffer pointer which saves the data to be sent.
<i>txSize</i>	Size of data to be sent in unit of byte.

UART HAL driver

18.2.3.26 uart_status_t UART_HAL_ReceiveDataPolling (uint32_t *baseAddr*, uint8_t * *rxBuff*, uint32_t *rxSize*)

This function only supports 8-bit transaction.

Parameters

<i>baseAddr</i>	UART module base address.
<i>rxBuff</i>	The buffer pointer which saves the data to be received.
<i>rxSize</i>	Size of data need to be received in unit of byte.

Returns

Whether the transaction is success or rx overrun.

18.2.3.27 static void UART_HAL_SetLoopCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function enables or disables the UART loopback operation.

Parameters

<i>baseAddr</i>	UART module base address.
<i>enable</i>	The UART loopback mode configuration, either disabled (false) or enabled (true).

18.2.3.28 static void UART_HAL_SetReceiverSource (uint32_t *baseAddr*, uart_receiver_source_t *source*) [inline], [static]

This function enables or disables the UART single-wire operation. In some UART baseAddrs it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	UART module base address.
<i>source</i>	The UART single-wire mode configuration.

18.2.3.29 static void UART_HAL_SetTransmitterDir (uint32_t *baseAddr*, uart_singlewire_txdir_t *direction*) [inline], [static]

This function configures the transmitter direction when the UART is configured for single-wire operation.

UART HAL driver

Parameters

<i>baseAddr</i>	UART module base address.
<i>direction</i>	The UART single-wire mode transmit direction configuration of type <code>uart-singlewire_txdir_t</code> (either <code>kUartSinglewireTxdirIn</code> or <code>kUartSinglewireTxdirOut</code>).

18.2.3.30 `uart_status_t UART_HAL_PutReceiverInStandbyMode (uint32_t baseAddr)`

This function, when called, places the UART receiver into standby mode. In some UART baseAddrs, there are conditions that must be met before placing Rx in standby mode. Before placing UART in standby, determine if receiver is set to wake on idle, and if receiver is already in idle state. NOTE: RWU should only be set with C1[WAKE] = 0 (wakeup on idle) if the channel is currently not idle. This can be determined by the S2[RAF] flag. If set to wake up FROM an IDLE event and the channel is already idle, it is possible that the UART will discard data because data must be received (or a LIN break detect) after an IDLE is detected before IDLE is allowed to be reasserted.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

Error code or `kStatus_UART_Success`.

18.2.3.31 `static void UART_HAL_PutReceiverInNormalMode (uint32_t baseAddr) [inline], [static]`

This function, when called, places the UART receiver into normal mode and out of standby mode.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

18.2.3.32 `static bool UART_HAL_IsReceiverInStandby (uint32_t baseAddr) [inline], [static]`

This function determines the state of the UART receiver. If it returns true, this means that the UART receiver is in standby mode; if it returns false, the UART receiver is in normal mode.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The UART receiver is in normal mode (false) or standby mode (true).

18.2.3.33 static void **UART_HAL_SetReceiverWakeupMethod** (*uint32_t baseAddr*, *uart_wakeup_method_t method*) [inline], [static]

This function configures the wakeup method of the UART receiver from standby mode. The options are idle-line wake or address-mark wake.

Parameters

<i>baseAddr</i>	UART module base address.
<i>method</i>	The UART receiver wakeup method options: kUartIdleLineWake - Idle-line wake or kUartAddrMarkWake - address-mark wake.

18.2.3.34 static *uart_wakeup_method_t* **UART_HAL_GetReceiverWakeupMethod** (*uint32_t baseAddr*) [inline], [static]

This function returns how the UART receiver is configured to wake from standby mode. The wake method options that can be returned are kUartIdleLineWake or kUartAddrMarkWake.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The UART receiver wakeup from standby method, false: kUartIdleLineWake (idle-line wake) or true: kUartAddrMarkWake (address-mark wake).

18.2.3.35 void **UART_HAL_ConfigIdleLineDetect** (*uint32_t baseAddr*, *uint8_t idleLine*, *uint8_t rxWakelineDetect*)

This function allows the user to configure the UART idle-line detect operation. There are two separate operations for the user to configure, the idle line bit-count start and the receive wake up affect on IDLE status bit. The user will pass in a structure of type *uart_idle_line_config_t*.

UART HAL driver

Parameters

<i>baseAddr</i>	UART module base address.
<i>idleLine</i>	Idle bit count start: 0 - after start bit (default), 1 - after stop bit
<i>rxWakeIdle-Detect</i>	Receiver Wake Up Idle Detect. IDLE status bit operation during receive standby. Controls whether idle character that wakes up receiver will also set IDLE status bit. 0 - IDLE status bit doesn't get set (default), 1 - IDLE status bit gets set

18.2.3.36 static void UART_HAL_SetBreakCharTransmitLength (uint32_t *baseAddr*, uart_break_char_length_t *length*) [inline], [static]

This function allows the user to configure the UART break character transmit length. Refer to the typedef `uart_break_char_length_t` for setting options. In some UART baseAddrs it is required that the transmitter be disabled before calling this function. This may be applied to all UARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	UART module base address.
<i>length</i>	The UART break character length setting of type <code>uart_break_char_length_t</code> , either a minimum 10-bit times or a minimum 13-bit times.

18.2.3.37 static void UART_HAL_SetBreakCharCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function allows the user to queue a UART break character to send. If true is passed into the function, then a break character is queued for transmission. A break character will continuously be queued until this function is called again when a false is passed into this function.

Parameters

<i>baseAddr</i>	UART module base address.
<i>enable</i>	If false, the UART normal/queue break character setting is disabled, which configures the UART for normal transmitter operation. If true, a break character is queued for transmission.

18.2.3.38 void UART_HAL_SetMatchAddress (uint32_t *baseAddr*, bool *matchAddrMode1*, bool *matchAddrMode2*, uint8_t *matchAddrValue1*, uint8_t *matchAddrValue2*)

(Note: Feature available on select UART baseAddrs)

The function allows the user to configure the UART match address control operation. The user has the option to enable the match address mode and to program the match address value. There are two match address modes, each with its own enable and programmable match address value.

Parameters

<i>baseAddr</i>	UART module base address.
<i>matchAddr-Mode1</i>	If true, this enables match address mode 1 (MAEN1), where false disables.
<i>matchAddr-Mode2</i>	If true, this enables match address mode 2 (MAEN2), where false disables.
<i>matchAddr-Value1</i>	The match address value to program for match address mode 1.
<i>matchAddr-Value2</i>	The match address value to program for match address mode 2.

18.2.3.39 **bool UART_HAL_GetStatusFlag (uint32_t *baseAddr*, uart_status_flag_t *statusFlag*)**

Parameters

<i>baseAddr</i>	UART module base address.
<i>statusFlag</i>	Status flag name.

Returns

Whether the current status flag is set(true) or not(false).

18.2.3.40 **static bool UART_HAL_IsTxDataRegEmpty (uint32_t *baseAddr*) [inline], [static]**

This function returns the state of the UART Transmit data register empty flag.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The status of Transmit data register empty flag, which is set when transmit buffer is empty.

18.2.3.41 static bool UART_HAL_IsTxComplete(uint32_t *baseAddr*) [inline], [static]

This function returns the state of the UART Transmission complete flag.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The status of Transmission complete flag, which is set when the transmitter is idle (transmission activity complete).

18.2.3.42 **static bool UART_HAL_IsRxDataRegFull(uint32_t *baseAddr*) [inline], [static]**

This function returns the state of the UART Receive data register full flag.

Parameters

<i>baseAddr</i>	UART module base address.
-----------------	---------------------------

Returns

The status of Receive data register full flag, which is set when the receive data buffer is full.

18.2.3.43 **uart_status_t UART_HAL_ClearStatusFlag(uint32_t *baseAddr*, uart_status_flag_t *statusFlag*)**

This function allows the user to clear an individual and specific UART status flag. Refer to structure definition `uart_status_flag_t` for list of status bits.

Parameters

<i>baseAddr</i>	UART module base address.
<i>statusFlag</i>	The desired UART status flag to clear.

Returns

An error code or `kStatus_UART_Success`.

18.3 UART Peripheral driver

18.3.1 Overview

The section describes the programming interface of the UART Peripheral driver. The UART peripheral driver transfers data to and from external devices on the Universal Asynchronous Receiver/Transmitter (UART) serial bus with a single function call.

18.3.2 UART Device structures

The driver uses an instantiation of the `uart_state_t` structure to maintain the current state of a particular UART instance module driver. This structure holds data used by the UART Peripheral driver to communicate between the transmit and receive transfer functions and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. Because the driver itself does not statically allocate memory, the caller provides memory for the driver state structure during initialization. The user is required to pass in the memory for the run-time state structure. The UART driver populates the structure members.

18.3.3 UART User configuration structures

The UART driver uses instances of the user configuration structure, `uart_user_config_t`, for the UART driver. As a result, the most common settings of the UART are configured with a single function call. Settings include: UART baud rate; UART parity mode: disabled (default), or even or odd; the number of stop bits; the number of bits per data word.

18.3.4 UART Initialization

1. To initialize the UART driver, call the `UART_DRV_Init()` function and pass the instance number of the UART peripheral you want to use, memory for the run-time state structure, and a pointer to the user configuration structure. For example, to use UART0 pass a value of 0 to the initialization function.
2. Then, pass the memory for the run-time state structure and, finally, pass a user configuration structure of the type `uart_user_config_t` as shown here:

```
// UART configuration structure
typedef struct UartUserConfig {
    uint32_t baudRate;
    uart_parity_mode_t parityMode;
    uart_stop_bit_count_t stopBitCount;
    uart_bit_count_per_char_t bitCountPerChar;
} uart_user_config_t;
```

Typically, the `uart_user_config_t` instantiation is configured as a 8-bit-character, no-parity, 1-stop-bit (8-n-1) with a 9600 bps baud rate. The `uart_user_config_t` instantiation can be easily modified to configure the UART Peripheral driver either to a different baud rate or character transfer features. This is an example code to set up a user UART configuration instantiation:

```
uart_user_config_t uartConfig;
uartConfig.baudRate = 9600;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
```

This example shows how to call the [UART_DRV_Init\(\)](#) function given the user configuration structure and the UART instance 0.

```
uint32_t uartInstance = 0;
uart_state_t uartState; // user provides memory for the driver state structure

UART_DRV_Init(uartInstance, &uartConfig, &uartState);
```

18.3.5 UART Transfers

The driver implements transmit and receive functions to transfer buffers of data. The driver also supports two different modes for transferring data: blocking and non-blocking.

The non-blocking transmit and receive functions include the [UART_DRV_SendData\(\)](#) and [UART_DRV_ReceiveData\(\)](#).

The blocking (async) transmit and receive functions include the [UART_DRV_SendDataBlocking\(\)](#) and [UART_DRV_ReceiveDataBlocking\(\)](#).

In all cases mentioned here, the functions are interrupt driven.

These code examples show how to use previously mentioned functions and assume that the UART module has been initialized as described previously in the Initialization Section.

For blocking transfer functions transmit and receive:

```
uint8_t sourceBuff[26] = {0}; // sourceBuff is populated with desired data
uint8_t readBuffer[10] = {0}; // readBuffer is populated with UART_DRV_ReceiveData function

uint32_t byteCount = sizeof(sourceBuff);
uint32_t rxRemainingSize = sizeof(readBuffer);

// for each use there, set timeout as "1"
// uartState is the run-time state. Pass in memory for this
// declared previously in the initialization chapter
UART_DRV_SendDataBlocking(&uartState, sourceBuff, byteCount, 1); // function won't
    return until transmit is complete
UART_DRV_ReceiveDataBlocking(&uartState, readBuffer, 1, timeoutValue); // function won't return until it receives all data
```

For non-blocking (async) transfer functions transmit and receive:

```
uint8_t *pTxBuff;
uint8_t rxBuff[10];
uint32_t txRemainingSize, rxRemainingSize;

// assume pTxBuff and txRemainingSize have been initialized
UART_DRV_SendData(&uartState, pTxBuff, txRemainingSize);

// now check on status of transmit and wait until done, the code can do something else and
```

UART Peripheral driver

```
// check back later, this is just an example
while (UART_DRV_GetTransmitStatus(&uartState, &bytesTransmittedCount) ==
    kStatus_UART_TxBusy);

// for receive, assume rxBuff is set up to receive data and rxRemainingSize is initialized
UART_DRV_ReceiveData(&uartState, rxBuff, rxRemainingSize);

// now check on status of receive and wait until done, the code can do something else and
// check back later, this is just an example
while (UART_DRV_GetReceiveStatus(&uartState, &bytesReceivedCount) ==
    kStatus_UART_RxBusy);
```

This section describes the programming interface of the UART Peripheral driver. The UART peripheral driver transfers data to and from external devices on the Universal Asynchronous Receiver/Transmitter (UART) serial bus with a single function call.

18.3.6 UART Device structures

The driver uses an instantiation of the `uart_state_t` structure to maintain the current state of a particular UART instance module driver. This structure holds data that is used by the UART Peripheral driver to communicate between the transmit and receive transfer functions and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. Because the driver itself does not statically allocate memory, the caller provides memory for the driver state structure during initialization by providing a structure. The UART driver populates the structure members.

18.3.7 UART User configuration structures

The UART driver uses instances of the user configuration structure `uart_user_config_t` for the UART driver. This enables configuration of the most common settings of the UART with a single function call. Settings include: UART baud rate; UART parity mode: disabled (default), or even or odd; the number of stop bits; the number of bits per data word.

18.3.8 UART Initialization

1. To initialize the UART driver, call the `UART_DRV_Init()` function and pass the instance number of the UART peripheral, memory for the run-time state structure, and a pointer to the user configuration structure. For example, to use UART0 pass a value of 0 to the initialization function.
2. Then, pass the memory for the run-time state structure.
3. Finally, pass a user configuration structure of the type `uart_user_config_t` as shown here:

```
// UART configuration structure
typedef struct UartUserConfig {
    uint32_t baudRate;
    uart_parity_mode_t parityMode;
    uart_stop_bit_count_t stopBitCount;
    uart_bit_count_per_char_t bitCountPerChar;
} uart_user_config_t;
```

Typically, the `uart_user_config_t` instantiation is configured as 8-bit-char, no-parity, 1-stop-bit (8-n-1) with a 9600 bps baud rate. The `uart_user_config_t` instantiation can be easily modified to configure the UART Peripheral driver either to a different baud rate or character transfer features. This is an example code to set up a user UART configuration instantiation:

```
uart_user_config_t uartConfig;
uartConfig.baudRate = 9600;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
```

This example shows how to call the `UART_DRV_Init()` given the user configuration structure and the UART instance 0.

```
uint32_t uartInstance = 0;
uart_state_t uartState; // user provides memory for the driver state structure

UART_DRV_Init(uartInstance, &uartConfig, &uartState);
```

18.3.9 UART Transfers

The driver implements transmit and receive functions to transfer buffers of data in blocking and non-blocking modes.

The non-blocking transmit and receive functions include the `UART_DRV_SendData()` and `UART_DRV_ReceiveData()`.

The blocking (async) transmit and receive functions include the `UART_DRV_SendDataBlocking()` and `UART_DRV_ReceiveDataBlocking()`.

In all cases mentioned here, the functions are interrupt driven.

This code examples show how to use the functions, assuming that the UART module has been initialized as described previously in the Initialization Section.

For blocking transfer functions transmit and receive:

```
uint8_t sourceBuff[26] ={0}; // sourceBuff can be filled out with desired data
uint8_t readBuffer[10] = {0}; // readBuffer gets filled with UART_DRV_ReceiveData function

uint32_t byteCount = sizeof(sourceBuff);
uint32_t rxRemainingSize = sizeof(readBuffer);

// for each use there, set timeout as "1"
// uartState is the run-time state. Pass in memory for this
// declared previously in the initialization chapter
UART_DRV_SendDataBlocking(&uartState, sourceBuff, byteCount, 1); // function won't
    return until transmit is complete
UART_DRV_ReceiveDataBlocking(&uartState, readBuffer, 1, timeoutValue); // function won't return until it receives all data
```

For non-blocking (async) transfer functions transmit and receive:

UART Peripheral driver

```
uint8_t *pTxBuff;
uint8_t rxBuff[10];
uint32_t txRemainingSize, rxRemainingSize;

// assume pTxBuff and txRemainingSize have been initialized
UART_DRV_SendData(&uartState, pTxBuff, txRemainingSize);

// now check on status of transmit and wait until done, the code can do something else and
// check back later, this is just an example
while (UART_DRV_GetTransmitStatus(&uartState, &bytesTransmittedCount) ==
      kStatus_UART_TxBusy);

// for receive, assume rxBuff is set up to receive data and rxRemainingSize is initialized
UART_DRV_ReceiveData(&uartState, rxBuff, rxRemainingSize);

// now check on status of receive and wait until done, the code can do something else and
// check back later, this is just an example
while (UART_DRV_GetReceiveStatus(&uartState, &bytesReceivedCount) ==
      kStatus_UART_RxBusy);
```

Data Structures

- struct [uart_state_t](#)
Runtime state of the UART driver. [More...](#)
- struct [uart_user_config_t](#)
User configuration structure for the UART driver. [More...](#)
- struct [uart_edma_state_t](#)
Runtime state structure for UART driver with EDMA. [More...](#)
- struct [uart_edma_user_config_t](#)
User configuration structure for the UART driver. [More...](#)

Typedefs

- [typedef void\(* uart_rx_callback_t \)](#)(uint32_t instance, void *uartState)
UART receive callback function type.

Variables

- [const uint32_t g_uartBaseAddr \[HW_UART_INSTANCE_COUNT\]](#)
Table of base addresses for UART instances.
- [const IRQn_Type g_uartRxTxIrqId \[HW_UART_INSTANCE_COUNT\]](#)
Table to save UART IRQ enumeration numbers defined in the CMSIS header file.
- [const uint32_t g_uartBaseAddr \[HW_UART_INSTANCE_COUNT\]](#)
Table of base addresses for UART instances.

UART Interrupt Driver

- [uart_status_t UART_DRV_Init](#) (uint32_t instance, [uart_state_t](#) *uartStatePtr, const [uart_user_config_t](#) *uartUserConfig)
Initializes an UART instance for operation.

- void **UART_DRV_Deinit** (uint32_t instance)

Shuts down the UART by disabling interrupts and the transmitter/receiver.
- **uart_rx_callback_t** **UART_DRV_InstallRxCallback** (uint32_t instance, **uart_rx_callback_t** function, uint8_t *rxBuff, void *callbackParam, bool alwaysEnableRxIrq)

Installs callback function for the UART receive.
- **uart_status_t** **UART_DRV_SendDataBlocking** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize, uint32_t timeout)

Sends (transmits) data out through the UART module using a blocking method.
- **uart_status_t** **UART_DRV_SendData** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize)

Sends (transmits) data through the UART module using a non-blocking method.
- **uart_status_t** **UART_DRV_GetTransmitStatus** (uint32_t instance, uint32_t *bytesRemaining)

Returns whether the previous UART transmit has finished.
- **uart_status_t** **UART_DRV_AbortSendingData** (uint32_t instance)

Terminates an asynchronous UART transmission early.
- **uart_status_t** **UART_DRV_ReceiveDataBlocking** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize, uint32_t timeout)

Gets (receives) data from the UART module using a blocking method.
- **uart_status_t** **UART_DRV_ReceiveData** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize)

Gets (receives) data from the UART module using a non-blocking method.
- **uart_status_t** **UART_DRV_GetReceiveStatus** (uint32_t instance, uint32_t *bytesRemaining)

Returns whether the previous UART receive is complete.
- **uart_status_t** **UART_DRV_AbortReceivingData** (uint32_t instance)

Terminates an asynchronous UART receive early.

UART EDMA Driver

- **uart_status_t** **UART_DRV_EdmaInit** (uint32_t instance, **uart_edma_state_t** *uartEdmaStatePtr, const **uart_edma_user_config_t** *uartUserConfig)

Initializes an UART instance to work with EDMA.
- void **UART_DRV_EdmaDeinit** (uint32_t instance)

Shuts down the UART.
- **uart_status_t** **UART_DRV_EdmaSendDataBlocking** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize, uint32_t timeout)

Sends (transmits) data out through the UART-EDMA module using a blocking method.
- **uart_status_t** **UART_DRV_EdmaSendData** (uint32_t instance, const uint8_t *txBuff, uint32_t txSize)

Sends (transmits) data through the UART-EDMA module using a non-blocking method.
- **uart_status_t** **UART_DRV_EdmaGetTransmitStatus** (uint32_t instance, uint32_t *bytesRemaining)

Returns whether the previous UART-EDMA transmit has finished.
- **uart_status_t** **UART_DRV_EdmaAbortSendingData** (uint32_t instance)

Terminates a non-blocking UART-EDMA transmission early.
- **uart_status_t** **UART_DRV_EdmaReceiveDataBlocking** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize, uint32_t timeout)

Gets (receives) data from the UART-EDMA module using a blocking method.
- **uart_status_t** **UART_DRV_EdmaReceiveData** (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize)

Gets (receives) data from the UART-EDMA module using a non-blocking method.
- **uart_status_t** **UART_DRV_EdmaGetReceiveStatus** (uint32_t instance, uint32_t *bytesRemaining)

Returns whether the previous UART-EDMA receive is complete.
- **uart_status_t** **UART_DRV_EdmaAbortReceivingData** (uint32_t instance)

Terminates a non-blocking UART-EDMA receive early.

18.3.10 Data Structure Documentation

18.3.10.1 struct uart_state_t

This structure holds data that are used by the UART peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress. The user passes in the memory for the run-time state structure and the UART driver fills out the members.

Data Fields

- `uint8_t txFifoEntryCount`
Number of data word entries in TX FIFO.
- `const uint8_t * txBuff`
The buffer of data being sent.
- `uint8_t * rxBuff`
The buffer of received data.
- `volatile size_t txSize`
The remaining number of bytes to be transmitted.
- `volatile size_t rxSize`
The remaining number of bytes to be received.
- `volatile bool isTxBusy`
True if there is an active transmit.
- `volatile bool isRxBusy`
True if there is an active receive.
- `volatile bool isTxBlocking`
True if transmit is blocking transaction.
- `volatile bool isRxBlocking`
True if receive is blocking transaction.
- `semaphore_t txIrqSync`
Used to wait for ISR to complete its TX business.
- `semaphore_t rxIrqSync`
Used to wait for ISR to complete its RX business.
- `uart_rx_callback_t rxCallback`
Callback to invoke after receiving byte.
- `void * rxCallbackParam`
Receive callback parameter pointer.

18.3.10.1.0.38 Field Documentation

- 18.3.10.1.0.38.1 `uint8_t uart_state_t::txFifoEntryCount`
- 18.3.10.1.0.38.2 `const uint8_t* uart_state_t::txBuff`
- 18.3.10.1.0.38.3 `uint8_t* uart_state_t::rxBuff`
- 18.3.10.1.0.38.4 `volatile size_t uart_state_t::txSize`
- 18.3.10.1.0.38.5 `volatile size_t uart_state_t::rxSize`
- 18.3.10.1.0.38.6 `volatile bool uart_state_t::isTxBusy`
- 18.3.10.1.0.38.7 `volatile bool uart_state_t::isRxBusy`
- 18.3.10.1.0.38.8 `volatile bool uart_state_t::isTxBlocking`
- 18.3.10.1.0.38.9 `volatile bool uart_state_t::isRxBlocking`
- 18.3.10.1.0.38.10 `semaphore_t uart_state_t::txIrqSync`
- 18.3.10.1.0.38.11 `semaphore_t uart_state_t::rxIrqSync`
- 18.3.10.1.0.38.12 `uart_rx_callback_t uart_state_t::rxCallback`
- 18.3.10.1.0.38.13 `void* uart_state_t::rxCallbackParam`

18.3.10.2 `struct uart_user_config_t`

Use an instance of this structure with the `UART_DRV_Init()` function. This enables configuration of the most common settings of the UART peripheral with a single function call. Settings include: UART baud rate, UART parity mode: disabled (default), or even or odd, the number of stop bits, and the number of bits per data word.

Data Fields

- `uint32_t baudRate`
UART baud rate.
- `uart_parity_mode_t parityMode`
parity mode, disabled (default), even, odd
- `uart_stop_bit_count_t stopBitCount`
number of stop bits, 1 stop bit (default) or 2 stop bits
- `uart_bit_count_per_char_t bitCountPerChar`
number of bits, 8-bit (default) or 9-bit in a word (up to 10-bits in some UART instances)

UART Peripheral driver

18.3.10.3 struct uart_edma_state_t

Data Fields

- volatile bool `isTxBusy`
True if there is an active transmit.
- volatile bool `isRxBusy`
True if there is an active receive.
- volatile bool `isTxBlocking`
True if transmit is blocking transaction.
- volatile bool `isRxBlocking`
True if receive is blocking transaction.
- `semaphore_t txIrqSync`
Used to wait for ISR to complete its TX business.
- `semaphore_t rxIrqSync`
Used to wait for ISR to complete its RX business.
- `edma_chn_state_t edmaUartTx`
Structure definition for the EDMA channel.
- `edma_chn_state_t edmaUartRx`
Structure definition for the EDMA channel.

18.3.10.3.0.39 Field Documentation

18.3.10.3.0.39.1 volatile bool uart_edma_state_t::isTxBusy

18.3.10.3.0.39.2 volatile bool uart_edma_state_t::isRxBusy

18.3.10.3.0.39.3 volatile bool uart_edma_state_t::isTxBlocking

18.3.10.3.0.39.4 volatile bool uart_edma_state_t::isRxBlocking

18.3.10.3.0.39.5 semaphore_t uart_edma_state_t::txIrqSync

18.3.10.3.0.39.6 semaphore_t uart_edma_state_t::rxIrqSync

18.3.10.4 struct uart_edma_user_config_t

Use an instance of this structure with the `UART_DRV_Init()` function. This enables configuration of the most common settings of the UART peripheral with a single function call. Settings include: UART baud rate, UART parity mode: disabled (default), or even or odd, the number of stop bits, and the number of bits per data word.

Data Fields

- `uint32_t baudRate`
UART baud rate.
- `uart_parity_mode_t parityMode`
parity mode, disabled (default), even, odd
- `uart_stop_bit_count_t stopBitCount`
number of stop bits, 1 stop bit (default) or 2 stop bits

- **uart_bit_count_per_char_t bitCountPerChar**
number of bits, 8-bit (default) or 9-bit in a word (up to 10-bits in some UART instances)

18.3.11 Function Documentation

18.3.11.1 **uart_status_t UART_DRV_Init (uint32_t instance, uart_state_t * uartStatePtr, const uart_user_config_t * uartUserConfig)**

This function initializes the run-time state structure to keep track of the on-going transfers, ungates the clock to the UART module, initializes the module to user-defined settings and default settings, configures the IRQ state structure and enables the module-level interrupt to the core, and enables the UART module transmitter and receiver. This example shows how to set up the `uart_state_t` and the `uart_user_config_t` parameters and how to call the `UART_DRV_Init` function by passing in these parameters:

```
uart_user_config_t uartConfig;
uartConfig.baudRate = 9600;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
uart_state_t uartState;
UART_DRV_Init(instance, &uartState, &uartConfig);
```

Parameters

<i>instance</i>	The UART instance number.
<i>uartStatePtr</i>	A pointer to the UART driver state structure memory. The user passes in the memory for this run-time state structure. The UART driver populates the members. The run-time state structure keeps track of the current transfer in progress.
<i>uartUserConfig</i>	The user configuration structure of type <code>uart_user_config_t</code> . The user populates the members of this structure and passes the pointer of this structure to this function.

Returns

An error code or `kStatus_UART_Success`.

18.3.11.2 **void UART_DRV_Deinit (uint32_t instance)**

This function disables the UART interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs).

UART Peripheral driver

Parameters

<i>instance</i>	The UART instance number.
-----------------	---------------------------

**18.3.11.3 `uart_rx_callback_t UART_DRV_InstallRxCallback (uint32_t instance,
 uart_rx_callback_t function, uint8_t * rxBuff, void * callbackParam, bool
 alwaysEnableRxIrq)`**

Note

Once a callback is installed, it bypasses the UART driver logic. So, the callback needs to handle the indexes of *rxBuff*, *rxSize*.

Parameters

<i>instance</i>	The UART instance number.
<i>function</i>	The UART receive callback function.
<i>rxBuff</i>	The receive buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive.
<i>callbackParam</i>	The UART receive callback parameter pointer.
<i>alwaysEnable-RxIrq</i>	Whether always enable Rx IRQ or not.

Returns

Former UART receive callback function pointer.

**18.3.11.4 `uart_status_t UART_DRV_SendDataBlocking (uint32_t instance, const uint8_t
 * txBuff, uint32_t txSize, uint32_t timeout)`**

A blocking (also known as synchronous) function means that the function does not return until the transmit is complete. This blocking function is used to send data through the UART port.

Parameters

<i>instance</i>	The UART instance number.
-----------------	---------------------------

<i>txBuff</i>	A pointer to the source buffer containing 8-bit data chars to send.
<i>txSize</i>	The number of bytes to send.
<i>timeout</i>	A timeout value for RTOS abstraction sync control in milliseconds (ms).

Returns

An error code or kStatus_UART_Success.

18.3.11.5 **uart_status_t UART_DRV_SendData (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*)**

A non-blocking (also known as synchronous) function means that the function returns immediately after initiating the transmit function. The application has to get the transmit status to see when the transmit is complete. In other words, after calling non-blocking (asynchronous) send function, the application must get the transmit status to check if transmit is complete. The asynchronous method of transmitting and receiving allows the UART to perform a full duplex operation (simultaneously transmit and receive).

Parameters

<i>instance</i>	The UART module base address.
<i>txBuff</i>	A pointer to the source buffer containing 8-bit data chars to send.
<i>txSize</i>	The number of bytes to send.

Returns

An error code or kStatus_UART_Success.

18.3.11.6 **uart_status_t UART_DRV_GetTransmitStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

When performing an async transmit, call this function to ascertain the state of the current transmission: in progress (or busy) or complete (success). If the transmission is still in progress, the user can obtain the number of words that have been transferred.

Parameters

UART Peripheral driver

<i>instance</i>	The UART module base address.
<i>bytes-Remaining</i>	A pointer to a value that is filled in with the number of bytes that are remaining in the active transfer.

Returns

The transmit status.

Return values

<i>kStatus_UART_Success</i>	The transmit has completed successfully.
<i>kStatus_UART_TxBusy</i>	The transmit is still in progress. <i>bytesTransmitted</i> is filled with the number of bytes which are transmitted up to that point.

18.3.11.7 **uart_status_t UART_DRV_AbortSendingData (uint32_t *instance*)**

During an async UART transmission, the user can terminate the transmission early if the transmission is still in progress.

Parameters

<i>instance</i>	The UART module base address.
-----------------	-------------------------------

Returns

Whether the aborting success or not.

Return values

<i>kStatus_UART_Success</i>	The transmit was successful.
<i>kStatus_UART_No-TransmitInProgress</i>	No transmission is currently in progress.

18.3.11.8 **uart_status_t UART_DRV_ReceiveDataBlocking (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*, uint32_t *timeout*)**

A blocking (also known as synchronous) function means that the function does not return until the receive is complete. This blocking function sends data through the UART port.

Parameters

<i>instance</i>	The UART module base address.
<i>rxBuff</i>	A pointer to the buffer containing 8-bit read data chars received.
<i>rxSize</i>	The number of bytes to receive.
<i>timeout</i>	A timeout value for RTOS abstraction sync control in milliseconds (ms).

Returns

An error code or kStatus_UART_Success.

18.3.11.9 **uart_status_t UART_DRV_ReceiveData (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*)**

A non-blocking (also known as synchronous) function means that the function returns immediately after initiating the receive function. The application has to get the receive status to see when the receive is complete. In other words, after calling non-blocking (asynchronous) get function, the application must get the receive status to check if receive is completed or not. The asynchronous method of transmitting and receiving allows the UART to perform a full duplex operation (simultaneously transmit and receive).

Parameters

<i>instance</i>	The UART module base address.
<i>rxBuff</i>	A pointer to the buffer containing 8-bit read data chars received.
<i>rxSize</i>	The number of bytes to receive.

Returns

An error code or kStatus_UART_Success.

18.3.11.10 **uart_status_t UART_DRV_GetReceiveStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

When performing an async receive, call this function to find out the state of the current receive progress: in progress (or busy) or complete (success). In addition, if the receive is still in progress, the user can obtain the number of words that have been currently received.

UART Peripheral driver

Parameters

<i>instance</i>	The UART module base address.
<i>bytesRemaining</i>	A pointer to a value that is filled in with the number of bytes which still need to be received in the active transfer.

Returns

The receive status.

Return values

<i>kStatus_UART_Success</i>	The receive has completed successfully.
<i>kStatus_UART_RxBusy</i>	The receive is still in progress. <i>bytesReceived</i> is filled with the number of bytes which are received up to that point.

18.3.11.11 **uart_status_t UART_DRV_AbortReceivingData (uint32_t *instance*)**

During an async UART receive, the user can terminate the receive early if the receive is still in progress.

Parameters

<i>instance</i>	The UART module base address.
-----------------	-------------------------------

Returns

Whether the aborting success or not.

Return values

<i>kStatus_UART_Success</i>	The receive was successful.
<i>kStatus_UART_NoTransmitInProgress</i>	No receive is currently in progress.

18.3.11.12 **uart_status_t UART_DRV_EdmaInit (uint32_t *instance*, uart_edma_state_t * *uartEdmaStatePtr*, const uart_edma_user_config_t * *uartUserConfig*)**

This function initializes the run-time state structure to keep track of the on-going transfers, ungates the clock to the UART module, initializes the module to user-defined settings and default settings, configures the IRQ state structure and enables the module-level interrupt to the core, and enables the UART module transmitter and receiver. This example shows how to set up the [uart_edma_state_t](#) and the [uart_user_config_t](#) parameters and how to call the [UART_DRV_EdmaInit](#) function by passing in these parameters:

```

uart_user_config_t uartConfig;
uartConfig.baudRate = 9600;
uartConfig.bitCountPerChar = kUart8BitsPerChar;
uartConfig.parityMode = kUartParityDisabled;
uartConfig.stopBitCount = kUartOneStopBit;
uart_edma_state_t uartEdmaState;
UART_DRV_EdmaInit(instance, &uartEdmaState, &uartConfig);

```

Parameters

<i>instance</i>	The UART instance number.
<i>uartEdma-StatePtr</i>	A pointer to the UART driver state structure memory. The user passes in the memory for the run-time state structure. The UART driver populates the members. This run-time state structure keeps track of the current transfer in progress.
<i>uartUserConfig</i>	The user configuration structure of type uart_user_config_t . The user populates the members of this structure and passes the pointer of this structure to this function.

Returns

An error code or [kStatus_UART_Success](#).

18.3.11.13 void [UART_DRV_EdmaDeinit](#) (uint32_t *instance*)

This function disables the UART-EDMA trigger and disables the transmitter and receiver.

Parameters

<i>instance</i>	The UART instance number.
-----------------	---------------------------

18.3.11.14 [uart_status_t \[UART_DRV_EdmaSendDataBlocking\]\(#\)](#) (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*, uint32_t *timeout*)

Parameters

<i>instance</i>	The UART instance number.
<i>txBuff</i>	A pointer to the source buffer containing 8-bit data chars to send.
<i>txSize</i>	The number of bytes to send.

UART Peripheral driver

<i>timeout</i>	A timeout value for RTOS abstraction sync control in milliseconds (ms).
----------------	---

Returns

An error code or kStatus_UART_Success.

18.3.11.15 **uart_status_t UART_DRV_EdmaSendData (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*)**

Parameters

<i>instance</i>	The UART module base address.
<i>txBuff</i>	A pointer to the source buffer containing 8-bit data chars to send.
<i>txSize</i>	The number of bytes to send.

Returns

An error code or kStatus_UART_Success.

18.3.11.16 **uart_status_t UART_DRV_EdmaGetTransmitStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

Parameters

<i>instance</i>	The UART module base address.
<i>bytesRemaining</i>	A pointer to a value that is populated with the number of bytes that are remaining in the active transfer.

Return values

<i>kStatus_UART_Success</i>	The transmit has completed successfully.
<i>kStatus_UART_TxBusy</i>	The transmit is still in progress. <i>bytesTransmitted</i> is filled with the number of bytes which are transmitted up to that point.

18.3.11.17 **uart_status_t UART_DRV_EdmaAbortSendingData (uint32_t *instance*)**

Parameters

<i>instance</i>	The UART module base address.
-----------------	-------------------------------

Return values

<i>kStatus_UART_Success</i>	The transmit was successful.
<i>kStatus_UART_NoTransmitInProgress</i>	No transmission is currently in progress.

18.3.11.18 **uart_status_t UART_DRV_EdmaReceiveDataBlocking (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*, uint32_t *timeout*)**

Parameters

<i>instance</i>	The UART module base address.
<i>rxBuff</i>	A pointer to the buffer containing 8-bit read data chars received.
<i>rxSize</i>	The number of bytes to receive.
<i>timeout</i>	A timeout value for RTOS abstraction sync control in milliseconds (ms).

Returns

An error code or *kStatus_UART_Success*.

18.3.11.19 **uart_status_t UART_DRV_EdmaReceiveData (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*)**

Parameters

<i>instance</i>	The UART module base address.
<i>rxBuff</i>	A pointer to the buffer containing 8-bit read data chars received.
<i>rxSize</i>	The number of bytes to receive.

Returns

An error code or *kStatus_UART_Success*.

18.3.11.20 **uart_status_t UART_DRV_EdmaGetReceiveStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

UART Peripheral driver

Parameters

<i>instance</i>	The UART module base address.
<i>bytesRemaining</i>	A pointer to a value that is populated with the number of bytes which still need to be received in the active transfer.

Return values

<i>kStatus_UART_Success</i>	The receive has completed successfully.
<i>kStatus_UART_RxBusy</i>	The receive is still in progress. <i>bytesReceived</i> is filled with the number of bytes which are received up to that point.

18.3.11.21 `uart_status_t UART_DRV_EdmaAbortReceivingData(uint32_t instance)`

Parameters

<i>instance</i>	The UART module base address.
-----------------	-------------------------------

Return values

<i>kStatus_UART_Success</i>	The receive was successful.
<i>kStatus_UART_NoTransmitInProgress</i>	No receive is currently in progress.

18.3.12 Variable Documentation

18.3.12.1 `const uint32_t g_uartBaseAddr[HW_UART_INSTANCE_COUNT]`

18.3.12.2 `const uint32_t g_uartBaseAddr[HW_UART_INSTANCE_COUNT]`

Chapter 19

Low Power Timer (LPTMR)

19.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Low Power Timer (LPTMR) block of Kinetis devices.

Modules

- [LPTMR HAL driver](#)
- [LPTMR Peripheral driver](#)

19.2 LPTMR HAL driver

19.2.1 Overview

This section describes the programming interface of the LPTMR HAL driver.

Enumerations

- enum `lptmr_pin_select_t` {
 `kLptmrPinSelectCmpOut` = 0x0U,
 `kLptmrPinSelectLptmrAlt1` = 0x1U,
 `kLptmrPinSelectLptmrAlt2` = 0x2U,
 `kLptmrPinSelectLptmrAlt3` = 0x3U }
 LPTMR pin selection.
- enum `lptmr_pin_polarity_t` {
 `kLptmrPinPolarityActiveHigh` = 0x0U,
 `kLptmrPinPolarityActiveLow` = 0x1U }
 LPTMR pin polarity, used while in pulse counter mode.
- enum `lptmr_timer_mode_t` {
 `kLptmrTimerModeTimeCounter` = 0x0U,
 `kLptmrTimerModePulseCounter` = 0x1U }
 LPTMR timer mode selection.
- enum `lptmr_prescaler_value_t` {
 `kLptmrPrescalerDivide2` = 0x0U,
 `kLptmrPrescalerDivide4GlichFiltch2` = 0x1U,
 `kLptmrPrescalerDivide8GlichFiltch4` = 0x2U,
 `kLptmrPrescalerDivide16GlichFiltch8` = 0x3U,
 `kLptmrPrescalerDivide32GlichFiltch16` = 0x4U,
 `kLptmrPrescalerDivide64GlichFiltch32` = 0x5U,
 `kLptmrPrescalerDivide128GlichFiltch64` = 0x6U,
 `kLptmrPrescalerDivide256GlichFiltch128` = 0x7U,
 `kLptmrPrescalerDivide512GlichFiltch256` = 0x8U,
 `kLptmrPrescalerDivide1024GlichFiltch512` = 0x9U,
 `kLptmrPrescalerDivide2048GlichFiltch1024` = 0xAU,
 `kLptmrPrescalerDivide4096GlichFiltch2048` = 0xBU,
 `kLptmrPrescalerDivide8192GlichFiltch4096` = 0xCU,
 `kLptmrPrescalerDivide16384GlichFiltch8192` = 0xDU,
 `kLptmrPrescalerDivide32768GlichFiltch16384` = 0xEU,
 `kLptmrPrescalerDivide65535GlichFiltch32768` = 0xFU }
 LPTMR prescaler value.
- enum `lptmr_status_t` {

```

kStatus_LPTMR_Success = 0x0U,
kStatus_LPTMR_NotInitialized = 0x1U,
kStatus_LPTMR_NullArgument = 0x2U,
kStatus_LPTMR_InvalidPrescalerValue = 0x3U,
kStatus_LPTMR_InvalidInTimeCounterMode = 0x4U,
kStatus_LPTMR_InvalidInPulseCounterMode = 0x5U,
kStatus_LPTMR_TcfNotSet = 0x6U,
kStatus_LPTMR_TimerPeriodUsTooSmall = 0x7U,
kStatus_LPTMR_TimerPeriodUsTooLarge = 0x8U }

```

LPTMR status return codes.

LPTMR HAL.

- static void [LPTMR_HAL_Enable](#) (uint32_t baseAddr)
Enables the LPTMR module operation.
- static void [LPTMR_HAL_Disable](#) (uint32_t baseAddr)
Disables the LPTMR module operation.
- static bool [LPTMR_HAL_IsEnabled](#) (uint32_t baseAddr)
Checks whether the LPTMR module is enabled.
- static void [LPTMR_HAL_ClearIntFlag](#) (uint32_t baseAddr)
Clears the LPTMR interrupt flag if set.
- static bool [LPTMR_HAL_IsIntPending](#) (uint32_t baseAddr)
Returns the current LPTMR interrupt flag.
- static void [LPTMR_HAL_SetIntCmd](#) (uint32_t baseAddr, bool enable)
Enables or disables the LPTMR interrupt.
- static bool [LPTMR_HAL_GetIntCmd](#) (uint32_t baseAddr)
Returns whether the LPTMR interrupt is enabled.
- static void [LPTMR_HAL_SetPinSelectMode](#) (uint32_t baseAddr, [lptmr_pin_select_t](#) pinSelect)
Selects the LPTMR pulse input pin select.
- static [lptmr_pin_select_t](#) [LPTMR_HAL_GetPinSelectMode](#) (uint32_t baseAddr)
Returns the LPTMR pulse input pin select.
- static void [LPTMR_HAL_SetPinPolarityMode](#) (uint32_t baseAddr, [lptmr_pin_polarity_t](#) pinPolarity)
Selects the LPTMR pulse input pin polarity.
- static [lptmr_pin_polarity_t](#) [LPTMR_HAL_GetPinPolarityMode](#) (uint32_t baseAddr)
Returns the LPTMR pulse input pin polarity.
- static void [LPTMR_HAL_SetFreeRunningCmd](#) (uint32_t baseAddr, bool enable)
Enables or disables the LPTMR free running.
- static bool [LPTMR_HAL_GetFreeRunningCmd](#) (uint32_t baseAddr)
Returns whether the LPTMR free running is enabled.
- static void [LPTMR_HAL_SetTimerModeMode](#) (uint32_t baseAddr, [lptmr_timer_mode_t](#) timerMode)
Selects the LPTMR working mode.
- static [lptmr_timer_mode_t](#) [LPTMR_HAL_GetTimerModeMode](#) (uint32_t baseAddr)
Returns the LPTMR working mode.
- static void [LPTMR_HAL_SetPrescalerValueMode](#) (uint32_t baseAddr, [lptmr_prescaler_value_t](#) prescaleValue)
Selects the LPTMR prescaler value.

LPTMR HAL driver

- static `lptmr_prescaler_value_t LPTMR_HAL_GetPrescalerValueMode (uint32_t baseAddr)`
Returns the LPTMR prescaler value.
- static void `LPTMR_HAL_SetPrescalerCmd (uint32_t baseAddr, bool enable)`
Enables or disables the LPTMR prescaler.
- static bool `LPTMR_HAL_GetPrescalerCmd (uint32_t baseAddr)`
Returns whether the LPTMR prescaler is enabled.
- static void `LPTMR_HAL_SetPrescalerClockSourceMode (uint32_t baseAddr, uint8_t prescalerClockSource)`
Selects the LPTMR clock source.
- static uint8_t `LPTMR_HAL_GetPrescalerClockSourceMode (uint32_t baseAddr)`
Gets the LPTMR clock source.
- static void `LPTMR_HAL_SetCompareValue (uint32_t baseAddr, uint32_t compareValue)`
Sets the LPTMR compare value.
- static uint32_t `LPTMR_HAL_GetCompareValue (uint32_t baseAddr)`
Gets the LPTMR compare value.
- static uint32_t `LPTMR_HAL_GetCounterValue (uint32_t baseAddr)`
Gets the LPTMR counter value.
- void `LPTMR_HAL_Init (uint32_t baseAddr)`
Restores the LPTMR module to reset state.

19.2.2 Enumeration Type Documentation

19.2.2.1 enum lptmr_pin_select_t

Enumerator

`kLptmrPinSelectCmpOut` Lptmr Pin is CMP0 output pin.

`kLptmrPinSelectLptmrAlt1` Lptmr Pin is LPTMR_ALT1 pin.

`kLptmrPinSelectLptmrAlt2` Lptmr Pin is LPTMR_ALT2 pin.

`kLptmrPinSelectLptmrAlt3` Lptmr Pin is LPTMR_ALT3 pin.

19.2.2.2 enum lptmr_pin_polarity_t

Enumerator

`kLptmrPinPolarityActiveHigh` Pulse Counter input source is active-high.

`kLptmrPinPolarityActiveLow` Pulse Counter input source is active-low.

19.2.2.3 enum lptmr_timer_mode_t

Enumerator

`kLptmrTimerModeTimeCounter` Time Counter mode.

`kLptmrTimerModePulseCounter` Pulse Counter mode.

19.2.2.4 enum lptmr_prescaler_value_t

Enumerator

kLptmrPrescalerDivide2 Prescaler divide 2, glitch filter invalid.
kLptmrPrescalerDivide4GlichFiltch2 Prescaler divide 4, glitch filter 2.
kLptmrPrescalerDivide8GlichFiltch4 Prescaler divide 8, glitch filter 4.
kLptmrPrescalerDivide16GlichFiltch8 Prescaler divide 16, glitch filter 8.
kLptmrPrescalerDivide32GlichFiltch16 Prescaler divide 32, glitch filter 16.
kLptmrPrescalerDivide64GlichFiltch32 Prescaler divide 64, glitch filter 32.
kLptmrPrescalerDivide128GlichFiltch64 Prescaler divide 128, glitch filter 64.
kLptmrPrescalerDivide256GlichFiltch128 Prescaler divide 256, glitch filter 128.
kLptmrPrescalerDivide512GlichFiltch256 Prescaler divide 512, glitch filter 256.
kLptmrPrescalerDivide1024GlichFiltch512 Prescaler divide 1024, glitch filter 512.
kLptmrPrescalerDivide2048GlichFiltch1024 Prescaler divide 2048 glitch filter 1024.
kLptmrPrescalerDivide4096GlichFiltch2048 Prescaler divide 4096, glitch filter 2048.
kLptmrPrescalerDivide8192GlichFiltch4096 Prescaler divide 8192, glitch filter 4096.
kLptmrPrescalerDivide16384GlichFiltch8192 Prescaler divide 16384, glitch filter 8192.
kLptmrPrescalerDivide32768GlichFiltch16384 Prescaler divide 32768, glitch filter 16384.
kLptmrPrescalerDivide65535GlichFiltch32768 Prescaler divide 65535, glitch filter 32768.

19.2.2.5 enum lptmr_status_t

Enumerator

kStatus_LPTMR_Success Succeed.
kStatus_LPTMR_NotInitialized LPTMR is not initialized yet.
kStatus_LPTMR_NullArgument Argument is NULL.
kStatus_LPTMR_InvalidPrescalerValue Value 0 is not valid in pulse counter mode.
kStatus_LPTMR_InvalidInTimeCounterMode Function can not called in time counter mode.
kStatus_LPTMR_InvalidInPulseCounterMode Function can not called in pulse counter mode.
kStatus_LPTMR_TcfNotSet If LPTMR is enabled, compare register can only altered when TCF is set.
kStatus_LPTMR_TimerPeriodUsTooSmall Timer period time is too small for current clock source.
kStatus_LPTMR_TimerPeriodUsTooLarge Timer period time is too large for current clock source.

19.2.3 Function Documentation

19.2.3.1 static void LPTMR_HAL_Enable (uint32_t baseAddr) [inline], [static]

LPTMR HAL driver

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

19.2.3.2 static void LPTMR_HAL_Disable (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

19.2.3.3 static bool LPTMR_HAL_IsEnabled (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Return values

<i>true</i>	LPTMR module is enabled.
<i>false</i>	LPTMR module is disabled.

19.2.3.4 static void LPTMR_HAL_ClearIntFlag (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

19.2.3.5 static bool LPTMR_HAL_IsIntPending (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address
-----------------	-----------------------------------

Return values

<i>true</i>	An interrupt is pending.
<i>false</i>	No interrupt is pending.

19.2.3.6 static void LPTMR_HAL_SetIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address
<i>enable</i>	Pass true to enable LPTMR interrupt

19.2.3.7 static bool LPTMR_HAL_GetIntCmd (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Return values

<i>true</i>	LPTMR interrupt is enabled.
<i>false</i>	LPTMR interrupt is disabled.

19.2.3.8 static void LPTMR_HAL_SetPinSelectMode (uint32_t *baseAddr*, lptmr_pin_select_t *pinSelect*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
<i>pinSelect</i>	Specifies LPTMR pulse input pin select, see lptmr_pin_select_t

19.2.3.9 static lptmr_pin_select_t LPTMR_HAL_GetPinSelectMode (uint32_t *baseAddr*) [inline], [static]

LPTMR HAL driver

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

LPTMR pulse input pin select, see [lptmr_pin_select_t](#)

19.2.3.10 static void LPTMR_HAL_SetPinPolarityMode (uint32_t *baseAddr*, lptmr_pin_polarity_t *pinPolarity*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
<i>pinPolarity</i>	Specifies LPTMR pulse input pin polarity, see lptmr_pin_polarity_t

19.2.3.11 static lptmr_pin_polarity_t LPTMR_HAL_GetPinPolarityMode (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

LPTMR pulse input pin polarity, see [lptmr_pin_polarity_t](#)

19.2.3.12 static void LPTMR_HAL_SetFreeRunningCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address
<i>enable</i>	Pass true to enable LPTMR free running

19.2.3.13 static bool LPTMR_HAL_GetFreeRunningCmd (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Return values

<i>true</i>	LPTMR free running is enabled.
<i>false</i>	LPTMR free running is disabled.

19.2.3.14 static void LPTMR_HAL_SetTimerModeMode (uint32_t *baseAddr*, lptmr_timer_mode_t *timerMode*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
<i>timerMode</i>	Specifies LPTMR working mode, see lptmr_timer_mode_t

19.2.3.15 static lptmr_timer_mode_t LPTMR_HAL_GetTimerModeMode (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

LPTMR working mode, see [lptmr_timer_mode_t](#)

19.2.3.16 static void LPTMR_HAL_SetPrescalerValueMode (uint32_t *baseAddr*, lptmr_prescaler_value_t *prescaleValue*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

LPTMR HAL driver

<i>prescaleValue</i>	Specifies LPTMR prescaler value, see lptmr_prescaler_value_t
----------------------	--

19.2.3.17 static lptmr_prescaler_value_t LPTMR_HAL_GetPrescalerValueMode (uint32_t baseAddr) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

LPTMR prescaler value, see [lptmr_prescaler_value_t](#)

19.2.3.18 static void LPTMR_HAL_SetPrescalerCmd (uint32_t baseAddr, bool enable) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address
<i>enable</i>	Pass true to enable LPTMR free running

19.2.3.19 static bool LPTMR_HAL_GetPrescalerCmd (uint32_t baseAddr) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Return values

<i>true</i>	LPTMR prescaler is enabled.
<i>false</i>	LPTMR prescaler is disabled.

19.2.3.20 static void LPTMR_HAL_SetPrescalerClockSourceMode (uint32_t baseAddr, uint8_t prescalerClockSource) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
<i>prescaler-ClockSource</i>	Specifies LPTMR clock source

19.2.3.21 static uint8_t LPTMR_HAL_GetPrescalerClockSourceMode (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

LPTMR clock source

19.2.3.22 static void LPTMR_HAL_SetCompareValue (uint32_t *baseAddr*, uint32_t *compareValue*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
<i>compareValue</i>	Specifies LPTMR compare value, less than 0xFFFFU

19.2.3.23 static uint32_t LPTMR_HAL_GetCompareValue (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

Current LPTMR compare value

19.2.3.24 static uint32_t LPTMR_HAL_GetCounterValue (uint32_t *baseAddr*) [inline], [static]

LPTMR HAL driver

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address.
-----------------	------------------------------------

Returns

Current LPTMR counter value

19.2.3.25 void LPTMR_HAL_Init (*uint32_t baseAddr*)

Parameters

<i>baseAddr</i>	The LPTMR peripheral base address
-----------------	-----------------------------------

19.3 LPTMR Peripheral driver

19.3.1 Overview

This section describes the programming interface of the LPTMR Peripheral driver. The LPTMR driver configures the LPTMR.

19.3.2 LPTMR Initialization

1. To initialize the LPTMR module, call the `LPTMR_DRV_Init()` function and pass in the user configuration data structure. This function automatically enables the LPTMR module and clock .
2. After the `LPTMR_DRV_Init()` function is called, call the `LPTMR_DRV_Start` to start the LPTMR.
3. To stop the LPTMR counter, call the `LPTMR_DRV_Stop`.

This is example code to configure the driver:

```
// Defines the LPTM user configuration structure . //
const lptmr_user_config_t config =
{
    .timerMode = kLptmrTimerModeTimerCounter, // timer counter is selected//
    .freeRunningEnable = false, // free running is disabled //
    .intEnable = true, // interrupt is enabled //
    .compareValue = 1024, // compare value is 1024 //
    .prescalerEnable = true, // prescaler is enabled //
    .prescalerClockSource = kLptmrPrescalerClockSourceLpo, // prescaler clock is LPO //
    .prescalerValue = kLptmrPrescalerDivide2, // prescaler divide is 2 //
};

// Initializes the LPTMR 0. //
LPTMR_DRV_Init(0,&config);

// Starts the LPTMR 0. //
LPTMR_DRV_Start(0);

// Stops LPTMR 0. //
LPTMR_DRV_Stop(0);

// Deinitializes LPTMR 0. //
LPTMR_DRV_Deinit(0);
```

19.3.3 LPTMR Interrupt

1. Enable the LPTMR interrupt. The LPTMR interrupt is enabled in the user configuration structure with the `lptmr_user_config_t.intEnable=true`.
2. Define the LPTMR IRQ function.

```
void LPTimer_IRQHandler()
{
    if(true == LPTMR_DRV_IsIntPending(0))
    {
        LPTMR_DRV_ClearIntFlag(0);
    }
    lptmrIsrAssertCount++;
}
```

LPTMR Peripheral driver

Data Structures

- struct [lptmr_user_config_t](#)
Data structure to initialize the LPTMR. [More...](#)
- struct [lptmr_state_t](#)
Internal driver state information. [More...](#)

Typedefs

- [typedef void\(* lptmr_callback_t \)\(void\)](#)
Defines a type of the user-defined callback function.

Variables

- [const uint32_t g_lptmrBaseAddr \[\]](#)
Table of base addresses for LPTMR instances.
- [const IRQn_Type g_lptmrIrqId \[HW_LPTMR_INSTANCE_COUNT\]](#)
Table to save LPTMR IRQ enumeration numbers defined in the CMSIS header file.

LPTMR Driver

- [lptmr_status_t LPTMR_DRV_Init \(uint32_t instance, const lptmr_user_config_t *userConfigPtr, lptmr_state_t *userStatePtr\)](#)
Initializes the LPTMR driver.
- [lptmr_status_t LPTMR_DRV_Deinit \(uint32_t instance\)](#)
De-initializes the LPTMR driver.
- [lptmr_status_t LPTMR_DRV_Start \(uint32_t instance\)](#)
Starts the LPTMR counter.
- [lptmr_status_t LPTMR_DRV_Stop \(uint32_t instance\)](#)
Stops the LPTMR counter.
- [lptmr_status_t LPTMR_DRV_SetTimerPeriodUs \(uint32_t instance, uint32_t us\)](#)
Configures the LPTMR timer period in microseconds.
- [uint32_t LPTMR_DRV_GetCurrentTimeUs \(uint32_t instance\)](#)
Gets the current LPTMR time in microseconds.
- [lptmr_status_t LPTMR_DRV_SetPulsePeriodCount \(uint32_t instance, uint32_t pulsePeriodCount\)](#)
Sets the pulse period value.
- [uint32_t LPTMR_DRV_GetCurrentPulseCount \(uint32_t instance\)](#)
Gets the current pulse count.
- [lptmr_status_t LPTMR_DRV_InstallCallback \(uint32_t instance, lptmr_callback_t userCallback\)](#)
Installs the user-defined callback in the LPTMR module.
- [void LPTMR_DRV_IRQHandler \(uint32_t instance\)](#)
Driver-defined ISR in the LPTMR module.

19.3.4 Data Structure Documentation

19.3.4.1 struct lptmr_user_config_t

This structure is used when initializing the LPTMR during the LPTMR_DRV_Init function call.

Data Fields

- `lptmr_timer_mode_t timerMode`
Timer counter mode or pulse counter mode.
- `lptmr_pin_select_t pinSelect`
LPTMR pulse input pin select.
- `lptmr_pin_polarity_t pinPolarity`
LPTMR pulse input pin polarity.
- `bool freeRunningEnable`
Free running configure.
- `bool prescalerEnable`
Prescaler enable configure.
- `clock_lptmr_src_t prescalerClockSource`
LPTMR clock source.
- `lptmr_prescaler_value_t prescalerValue`
Prescaler value.
- `bool isInterruptEnabled`
Timer interrupt 0-disable/1-enable.

19.3.4.1.0.40 Field Documentation

19.3.4.1.0.40.1 bool lptmr_user_config_t::freeRunningEnable

True means enable free running

19.3.4.1.0.40.2 bool lptmr_user_config_t::prescalerEnable

True means enable prescaler

19.3.4.2 struct lptmr_state_t

The contents of this structure are internal to the driver and should not be modified by users. Contents of the structure are subject to change in future releases.

Data Fields

- `lptmr_callback_t userCallbackFunc`
Callback function that is executed in ISR.

LPTMR Peripheral driver

19.3.4.2.0.41 Field Documentation

19.3.4.2.0.41.1 `lptmr_callback_t lptmr_state_t::userCallbackFunc`

19.3.5 Function Documentation

19.3.5.1 `lptmr_status_t LPTMR_DRV_Init (uint32_t instance, const lptmr_user_config_t * userConfigPtr, lptmr_state_t * userStatePtr)`

This function initializes the LPTMR. The LPTMR can be initialized as a time counter or pulse counter, which is determined by the timerMode in the `lptmr_user_config_t`. pinSelect and pinPolarity do not need to be configured while working as a time counter.

Parameters

<code>instance</code>	The LPTMR peripheral instance number.
<code>userConfigPtr</code>	The pointer to the LPTMR user configure structure, see lptmr_user_config_t .
<code>userStatePtr</code>	The pointer to the structure of the context memory, see lptmr_state_t .

Returns

`kStatus_LPTMR_Success` means succeed, otherwise means failed.

19.3.5.2 `lptmr_status_t LPTMR_DRV_Deinit (uint32_t instance)`

This function de-initializes the LPTMR. It disables the interrupt and turns off the LPTMR clock.

Parameters

<code>instance</code>	The LPTMR peripheral instance number.
-----------------------	---------------------------------------

Returns

`kStatus_LPTMR_Success` means succeed, otherwise means failed.

19.3.5.3 `lptmr_status_t LPTMR_DRV_Start (uint32_t instance)`

This function starts the LPTMR counter. Ensure that all necessary configurations are set before calling this function.

Parameters

<i>instance</i>	The LPTMR peripheral instance number.
-----------------	---------------------------------------

Returns

kStatus_LPTMR_Success means success. Otherwise, means failure.

19.3.5.4 lptmr_status_t LPTMR_DRV_Stop (uint32_t *instance*)

This function stops the LPTMR counter.

Parameters

<i>instance</i>	The LPTMR peripheral instance number.
-----------------	---------------------------------------

Returns

kStatus_LPTMR_Success means success. Otherwise, means failure.

19.3.5.5 lptmr_status_t LPTMR_DRV_SetTimerPeriodUs (uint32_t *instance*, uint32_t *us*)

This function configures the LPTMR time period while the LPTMR is working as a time counter. After the time period in microseconds, the callback function is called. This function cannot be called while the LPTMR is working as a pulse counter. The value in microseconds (*us*) should be integer multiple of the clock source time slice. If the clock source is 1 kHz, then both 2000 us and 3000 us are valid while 2500 us gets the same result as the 2000 us, because 2500 us cannot be generated in 1 kHz clock source.

Parameters

<i>instance</i>	The LPTMR peripheral instance number.
<i>us</i>	time period in microseconds.

Returns

kStatus_LPTMR_Success means success. Otherwise, means failure.

19.3.5.6 uint32_t LPTMR_DRV_GetCurrentTimeUs (uint32_t *instance*)

This function gets the current time while operating as a time counter. This function cannot be called while operating as a pulse counter.

LPTMR Peripheral driver

Parameters

<i>instance</i>	The LPTMR peripheral instance number.
-----------------	---------------------------------------

Returns

current time in microsecond unit.

19.3.5.7 lptmr_status_t LPTMR_DRV_SetPulsePeriodCount (uint32_t *instance*, uint32_t *pulsePeriodCount*)

This function configures the pulse period of the LPTMR while working as a pulse counter. After the count of pulsePeriodValue pulse is captured, the callback function is called. This function cannot be called while operating as a time counter.

Parameters

<i>instance</i>	The LPTMR peripheral instance number.
<i>pulsePeriod-Count</i>	pulse period value.

Returns

kStatus_LPTMR_Success means success. Otherwise, means failure.

19.3.5.8 uint32_t LPTMR_DRV_GetCurrentPulseCount (uint32_t *instance*)

This function gets the current pulse count captured on the pulse input pin. This function cannot be called while operating as a time counter.

Parameters

<i>instance</i>	The LPTMR peripheral instance number.
-----------------	---------------------------------------

Returns

pulse count captured on the pulse input pin.

19.3.5.9 lptmr_status_t LPTMR_DRV_InstallCallback (uint32_t *instance*, lptmr_callback_t *userCallback*)

This function installs the user-defined callback in the LPTMR module. When an LPTMR interrupt request is served, the callback is executed inside the ISR.

Parameters

<i>instance</i>	LPTMR instance ID.
<i>userCallback</i>	User-defined callback function.

Returns

kStatus_LPTMR_Success means success. Otherwise, means failure.

19.3.5.10 void LPTMR_DRV_IRQHandler (uint32_t *instance*)

This function is the driver-defined ISR in LPTMR module. It includes the process for interrupt mode defined by driver. Currently, it is called inside the system-defined ISR.

Parameters

<i>instance</i>	LPTMR instance ID.
-----------------	--------------------

19.3.6 Variable Documentation

19.3.6.1 const uint32_t g_lptmrBaseAddr[]

19.3.6.2 const IRQn_Type g_lptmrlrqId[HW_LPTMR_INSTANCE_COUNT]

Chapter 20

Low Power Universal Asynchronous Receiver/Transmitter (LPUART)

20.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Low Power Universal Asynchronous Receiver/Transmitter (LPUART) block of Kinetis devices.

Modules

- LPUART HAL driver
- LPUART Peripheral driver
- LPUART Type Definitions

LPUART HAL driver

20.2 LPUART HAL driver

20.2.1 Overview

This section describes the programming interface of the LPUART HAL driver.

Data Structures

- struct `lpuart_idle_line_config_t`
Structure for idle line configuration settings. [More...](#)

Enumerations

- enum `lpuart_status_t`
Error codes for the LPUART driver.
- enum `lpuart_stop_bit_count_t` {
 `kLpuartOneStopBit` = 0x0U,
 `kLpuartTwoStopBit` = 0x1U }
LPUART number of stop bits.
- enum `lpuart_parity_mode_t` {
 `kLpuartParityDisabled` = 0x0U,
 `kLpuartParityEven` = 0x2U,
 `kLpuartParityOdd` = 0x3U }
LPUART parity mode.
- enum `lpuart_bit_count_per_char_t` {
 `kLpuart8BitsPerChar` = 0x0U,
 `kLpuart9BitsPerChar` = 0x1U,
 `kLpuart10BitsPerChar` = 0x2U }
LPUART number of bits in a character.
- enum `lpuart_operation_config_t` {
 `kLpuartOperates` = 0x0U,
 `kLpuartStops` = 0x1U }
LPUART operation configuration constants.
- enum `lpuart_wakeup_method_t` {
 `kLpuartIdleLineWake` = 0x0U,
 `kLpuartAddrMarkWake` = 0x1U }
LPUART wakeup from standby method constants.
- enum `lpuart_idle_line_select_t` {
 `kLpuartIdleLineAfterStartBit` = 0x0U,
 `kLpuartIdleLineAfterStopBit` = 0x1U }
LPUART idle line detect selection types.
- enum `lpuart_break_char_length_t` {
 `kLpuartBreakChar10BitMinimum` = 0x0U,
 `kLpuartBreakChar13BitMinimum` = 0x1U }
LPUART break character length settings for transmit/detect.

- enum `lpuart_singlewire_txdirection_t` {

 `kLpuartSinglewireTxdirIn` = 0x0U,

 `kLpuartSinglewireTxdirOut` = 0x1U }

LPUART single-wire mode TX direction.
- enum `lpuart_match_config_t` {

 `kLpuartAddressMatchWakeup` = 0x0U,

 `kLpuartIdleMatchWakeup` = 0x1U,

 `kLpuartMatchOnAndMatchOff` = 0x2U,

 `kLpuartEnablesRwuOnDataMatch` = 0x3U }

LPUART Configures the match addressing mode used.
- enum `lpuart_ir_tx_pulsewidth_t` {

 `kLpuartIrThreeSixteenthsWidth` = 0x0U,

 `kLpuartIrOneSixteenthWidth` = 0x1U,

 `kLpuartIrOneThirtysecondsWidth` = 0x2U,

 `kLpuartIrOneFourthWidth` = 0x3U }

LPUART infra-red transmitter pulse width options.
- enum `lpuart_idle_char_t` {

 `kLpuart_1_IdleChar` = 0x0U,

 `kLpuart_2_IdleChar` = 0x1U,

 `kLpuart_4_IdleChar` = 0x2U,

 `kLpuart_8_IdleChar` = 0x3U,

 `kLpuart_16_IdleChar` = 0x4U,

 `kLpuart_32_IdleChar` = 0x5U,

 `kLpuart_64_IdleChar` = 0x6U,

 `kLpuart_128_IdleChar` = 0x7U }

LPUART Configures the number of idle characters that must be received before the IDLE flag is set.
- enum `lpuart_cts_source_t` {

 `kLpuartCtsSourcePin` = 0x0U,

 `kLpuartCtsSourceInvertedReceiverMatch` = 0x1U }

LPUART Transmits the CTS Configuration.
- enum `lpuart_cts_config_t` {

 `kLpuartCtsSampledOnEachChar` = 0x0U,

 `kLpuartCtsSampledOnIdle` = 0x1U }

LPUART Transmits CTS Source. Configures if the CTS state is checked at the start of each character or only when the transmitter is idle.
- enum `lpuart_status_flag_t` {

 `kLpuartTxDataRegEmpty` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_TDRE,

 `kLpuartTxComplete` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_TC,

 `kLpuartRxDataRegFull` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_RDRF,

 `kLpuartIdleLineDetect` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_IDLE,

 `kLpuartRxOverrun` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_O-

LPUART HAL driver

- R,
- `kLpuartNoiseDetect` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_NF,
`kLpuartFrameErr` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_FE,
`kLpuartParityErr` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_PF,
`kLpuartLineBreakDetect` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_LBKDE,
`kLpuartRxActiveEdgeDetect` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_RXEDGIF,
`kLpuartRxActive` = LPUART_STAT_REG_ID << LPUART_SHIFT | BP_LPUART_STAT_RAF,
`kLpuartNoiseInCurrentWord` = LPUART_DATA_REG_ID << LPUART_SHIFT | BP_LPUART_DATA_NOISY,
`kLpuartParityErrInCurrentWord` = LPUART_DATA_REG_ID << LPUART_SHIFT | BP_LPUART_DATA_PARITYE }
- LPUART status flags.*
- enum `lpuart_interrupt_t` {
 `kLpuartIntLinBreakDetect` = LPUART_BAUD_REG_ID << LPUART_SHIFT | BP_LPUART_BAUD_LBKDIE,
 `kLpuartIntRxActiveEdge` = LPUART_BAUD_REG_ID << LPUART_SHIFT | BP_LPUART_BAUD_RXEDGIE,
 `kLpuartIntTxDataRegEmpty` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_TIE,
 `kLpuartIntTxComplete` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_TCIE,
 `kLpuartIntRxDataRegFull` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_RIE,
 `kLpuartIntIdleLine` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_IIE,
 `kLpuartIntRxOverrun` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_ORIE,
 `kLpuartIntNoiseErrFlag` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_NEIE,
 `kLpuartIntFrameErrFlag` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_FEIE,
 `kLpuartIntParityErrFlag` = LPUART_CTRL_REG_ID << LPUART_SHIFT | BP_LPUART_CTRL_PEIE }
- LPUART interrupt configuration structure, default settings are 0 (disabled)*

LPUART Common Configurations

- void `LPUART_HAL_Init` (uint32_t baseAddr)
Initializes the LPUART controller to known state.
- static void `LPUART_HAL_SetTransmitterCmd` (uint32_t baseAddr, bool enable)
Enable/Disable the LPUART transmitter.

- static bool **LPUART_HAL_GetTransmitterCmd** (uint32_t baseAddr)
Gets the LPUART transmitter enabled/disabled configuration.
- static void **LPUART_HAL_SetReceiverCmd** (uint32_t baseAddr, bool enable)
Enable/Disable the LPUART receiver.
- static bool **LPUART_HAL_GetReceiverCmd** (uint32_t baseAddr)
Gets the LPUART receiver enabled/disabled configuration.
- lpuart_status_t **LPUART_HAL_SetBaudRate** (uint32_t baseAddr, uint32_t sourceClockInHz, uint32_t desiredBaudRate)
Configures the LPUART baud rate.
- static void **LPUART_HAL_SetBaudRateDivisor** (uint32_t baseAddr, uint32_t baudRateDivisor)
Sets the LPUART baud rate modulo divisor.
- void **LPUART_HAL_SetBitCountPerChar** (uint32_t baseAddr, lpuart_bit_count_per_char_t bitCountPerChar)
Configures the number of bits per character in the LPUART controller.
- void **LPUART_HAL_SetParityMode** (uint32_t baseAddr, lpuart_parity_mode_t parityModeType)
Configures parity mode in the LPUART controller.
- static void **LPUART_HAL_SetStopBitCount** (uint32_t baseAddr, lpuart_stop_bit_count_t stopBitCount)
Configures the number of stop bits in the LPUART controller.
- static void **LPUART_HAL_SetTxInversionCmd** (uint32_t baseAddr, bool enable)
Configures the transmit and receive inversion control in the LPUART controller.
- static void **LPUART_HAL_SetRxInversionCmd** (uint32_t baseAddr, bool enable)
Configures the transmit and receive inversion control in the LPUART controller.
- static uint32_t **LPUART_HAL_GetDataRegAddr** (uint32_t baseAddr)
Get LPUART tx/rx data register address.

LPUART Interrupts and DMA

- void **LPUART_HAL_SetIntMode** (uint32_t baseAddr, lpuart_interrupt_t interrupt, bool enable)
Configures the LPUART module interrupts to enable/disable various interrupt sources.
- bool **LPUART_HAL.GetIntMode** (uint32_t baseAddr, lpuart_interrupt_t interrupt)
Returns whether the LPUART module interrupts is enabled/disabled.
- static void **LPUART_HAL_SetTxDataRegEmptyIntCmd** (uint32_t baseAddr, bool enable)
Enable/Disable the transmission_complete_interrupt.
- static bool **LPUART_HAL_GetTxDataRegEmptyIntCmd** (uint32_t baseAddr)
Gets the configuration of the transmission_data_register_empty_interrupt enable setting.
- static void **LPUART_HAL_SetRxDataRegFullIntCmd** (uint32_t baseAddr, bool enable)
Enables the rx_data_register_full_interrupt.
- static bool **LPUART_HAL_GetRxDataRegFullIntCmd** (uint32_t baseAddr)
Gets the configuration of the rx_data_register_full_interrupt enable.

LPUART Transfer Functions

- static void **LPUART_HAL_Putchar** (uint32_t baseAddr, uint8_t data)
Sends the LPUART 8-bit character.
- void **LPUART_HAL_Putchar9** (uint32_t baseAddr, uint16_t data)
Sends the LPUART 9-bit character.
- lpuart_status_t **LPUART_HAL_Putchar10** (uint32_t baseAddr, uint16_t data)

LPUART HAL driver

Sends the LPUART 10-bit character (Note: Feature available on select LPUART instances).

- static void **LPUART_HAL_Getchar** (uint32_t baseAddr, uint8_t *readData)
 Gets the LPUART 8-bit character.
- void **LPUART_HAL_Getchar9** (uint32_t baseAddr, uint16_t *readData)
 Gets the LPUART 9-bit character.
- void **LPUART_HAL_Getchar10** (uint32_t baseAddr, uint16_t *readData)
 Gets the LPUART 10-bit character.
- void **LPUART_HAL_SendDataPolling** (uint32_t baseAddr, const uint8_t *txBuff, uint32_t txSize)
 Send out multiple bytes of data using polling method.
- **Ipuart_status_t LPUART_HAL_ReceiveDataPolling** (uint32_t baseAddr, uint8_t *rxBuff, uint32_t rxSize)
 Receive multiple bytes of data using polling method.
- static void **LPUART_HAL_SetIdleChar** (uint32_t baseAddr, **Ipuart_idle_char_t** idleConfig)
 Configures the number of idle characters that must be received before the IDLE flag is set.
- static **Ipuart_idle_char_t LPUART_HAL_GetIdleChar** (uint32_t baseAddr)
 Gets the configuration of the number of idle characters that must be received before the IDLE flag is set.
- static bool **LPUART_HAL_IsCurrentDataWithNoise** (uint32_t baseAddr)
 Checks whether the current data word was received with noise.
- static bool **LPUART_HAL_IsCurrentDataWithFrameError** (uint32_t baseAddr)
 Checks whether the current data word was received with frame error.
- static void **LPUART_HAL_SetTxSpecialChar** (uint32_t baseAddr, uint8_t specialChar)
 Set this bit to indicate a break or idle character is to be transmitted instead of the contents in DATA[T9:T0].
- static bool **LPUART_HAL_IsCurrentDataWithParityError** (uint32_t baseAddr)
 Checks whether the current data word was received with parity error.
- static bool **LPUART_HAL_IsReceiveBufferEmpty** (uint32_t baseAddr)
 Checks whether the receive buffer is empty.
- static bool **LPUART_HAL_WasPreviousReceiverStateIdle** (uint32_t baseAddr)
 Checks whether the previous BUS state was idle before this byte is received.

LPUART Special Feature Configurations

- static void **LPUART_HAL_SetWaitModeOperation** (uint32_t baseAddr, **Ipuart_operation_config_t** mode)
 Configures the LPUART operation in wait mode (operates or stops operations in wait mode).
- static **Ipuart_operation_config_t LPUART_HAL_GetWaitModeOperation** (uint32_t baseAddr)
 Gets the LPUART operation in wait mode (operates or stops operations in wait mode).
- void **LPUART_HAL_SetLoopbackCmd** (uint32_t baseAddr, bool enable)
 Configures the LPUART loopback operation (enable/disable loopback operation)
- void **LPUART_HAL_SetSingleWireCmd** (uint32_t baseAddr, bool enable)
 Configures the LPUART single-wire operation (enable/disable single-wire mode)
- static void **LPUART_HAL_SetTxdirInSinglewireMode** (uint32_t baseAddr, **Ipuart_singlewire_txdir_t** direction)
 Configures the LPUART transmit direction while in single-wire mode.
- **Ipuart_status_t LPUART_HAL_SetReceiverInStandbyMode** (uint32_t baseAddr)
 Places the LPUART receiver in standby mode.
- static void **LPUART_HAL_PutReceiverInNormalMode** (uint32_t baseAddr)
 Places the LPUART receiver in a normal mode (disable standby mode operation).
- static bool **LPUART_HAL_IsReceiverInStandby** (uint32_t baseAddr)

- Checks whether the LPUART receiver is in a standby mode.
- static void **LPUART_HAL_SetReceiverWakeupMode** (uint32_t baseAddr, lpuart_wakeup_method_t method)

LPUART receiver wakeup method (idle line or addr-mark) from standby mode.
- static lpuart_wakeup_method_t **LPUART_HAL_GetReceiverWakeupMode** (uint32_t baseAddr)

Gets the LPUART receiver wakeup method (idle line or addr-mark) from standby mode.
- void **LPUART_HAL_SetIdleLineDetect** (uint32_t baseAddr, const lpuart_idle_line_config_t *config)

LPUART idle-line detect operation configuration (idle line bit-count start and wake up affect on IDLE status bit).
- static void **LPUART_HAL_SetBreakCharTransmitLength** (uint32_t baseAddr, lpuart_break_char_length_t length)

LPUART break character transmit length configuration.
- static void **LPUART_HAL_SetBreakCharDetectLength** (uint32_t baseAddr, lpuart_break_char_length_t length)

LPUART break character detect length configuration.
- static void **LPUART_HAL_QueueBreakCharToSend** (uint32_t baseAddr, bool enable)

LPUART transmit sends break character configuration.
- static void **LPUART_HAL_SetMatchAddressMode** (uint32_t baseAddr, lpuart_match_config_t config)

LPUART configures match address mode control.
- void **LPUART_HAL_SetMatchAddressReg1** (uint32_t baseAddr, bool enable, uint8_t value)

Configures address match register 1.
- void **LPUART_HAL_SetMatchAddressReg2** (uint32_t baseAddr, bool enable, uint8_t value)

Configures address match register 2.
- static void **LPUART_HAL_SetSendMsbFirstCmd** (uint32_t baseAddr, bool enable)

LPUART sends the MSB first configuration.
- static void **LPUART_HAL_SetReceiveResyncCmd** (uint32_t baseAddr, bool enable)

LPUART enable/disable re-sync of received data configuration.

LPUART Status Flags

- bool **LPUART_HAL_GetStatusFlag** (uint32_t baseAddr, lpuart_status_flag_t statusFlag)

LPUART get status flag.
- static bool **LPUART_HAL_IsTxDataRegEmpty** (uint32_t baseAddr)

Gets the LPUART Transmit data register empty flag.
- static bool **LPUART_HAL_IsRxDataRegFull** (uint32_t baseAddr)

Gets the LPUART receive data register full flag.
- static bool **LPUART_HAL_IsTxComplete** (uint32_t baseAddr)

Gets the LPUART transmission complete flag.
- lpuart_status_t **LPUART_HAL_ClearStatusFlag** (uint32_t baseAddr, lpuart_status_flag_t statusFlag)

LPUART clears an individual status flag (see lpuart_status_flag_t for list of status bits).

20.2.2 Data Structure Documentation

20.2.2.1 struct lpuart_idle_line_config_t

Data Fields

- unsigned `idleLineType`: 1
ILT, Idle bit count start: 0 - after start bit (default), 1 - after stop bit.
- unsigned `rxWakeIdleDetect`: 1
RWUID, Receiver Wake Up Idle Detect.

20.2.2.1.0.42 Field Documentation

20.2.2.1.0.42.1 `unsigned lpuart_idle_line_config_t::rxWakeIdleDetect`

IDLE status bit operation during receive standbyControls whether idle character that wakes up receiver will also set IDLE status bit 0 - IDLE status bit doesn't get set (default), 1 - IDLE status bit gets set

20.2.3 Enumeration Type Documentation

20.2.3.1 enum lpuart_status_t

20.2.3.2 enum lpuart_stop_bit_count_t

Enumerator

- kLpuartOneStopBit* one stop bit
kLpuartTwoStopBit two stop bits

20.2.3.3 enum lpuart_parity_mode_t

Enumerator

- kLpuartParityDisabled* parity disabled
kLpuartParityEven parity enabled, type even, bit setting: PE|PT = 10
kLpuartParityOdd parity enabled, type odd, bit setting: PE|PT = 11

20.2.3.4 enum lpuart_bit_count_per_char_t

Enumerator

- kLpuart8BitsPerChar* 8-bit data characters
kLpuart9BitsPerChar 9-bit data characters
kLpuart10BitsPerChar 10-bit data characters

20.2.3.5 enum lpuart_operation_config_t

Enumerator

kLpuartOperates LPUART continues to operate normally.

kLpuartStops LPUART stops operation.

20.2.3.6 enum lpuart_wakeup_method_t

Enumerator

kLpuartIdleLineWake Idle-line wakes the LPUART receiver from standby.

kLpuartAddrMarkWake Addr-mark wakes LPUART receiver from standby.

20.2.3.7 enum lpuart_idle_line_select_t

Enumerator

kLpuartIdleLineAfterStartBit LPUART idle character bit count start after start bit.

kLpuartIdleLineAfterStopBit LPUART idle character bit count start after stop bit.

20.2.3.8 enum lpuart_break_char_length_t

The actual maximum bit times may vary depending on the LPUART instance.

Enumerator

kLpuartBreakChar10BitMinimum LPUART break char length 10 bit times (if M = 0, SBNS = 0 or 11 (if M = 1, SBNS = 0 or M = 0, SBNS = 1) or 12 (if M = 1, SBNS = 1 or M10 = 1, SNBS = 0) or 13 (if M10 = 1, SNBS = 1))

kLpuartBreakChar13BitMinimum LPUART break char length 13 bit times (if M = 0, SBNS = 0 or M10 = 0, SBNS = 1) or 14 (if M = 1, SBNS = 0 or M = 1, SBNS = 1) or 15 (if M10 = 1, SBNS = 1 or M10 = 1, SNBS = 0)

20.2.3.9 enum lpuart_singlewire_txdir_t

Enumerator

kLpuartSinglewireTxdirIn LPUART Single Wire mode TXDIR input.

kLpuartSinglewireTxdirOut LPUART Single Wire mode TXDIR output.

LPUART HAL driver

20.2.3.10 enum lpuart_match_config_t

Enumerator

kLpuartAddressMatchWakeup Address Match Wakeup.

kLpuartIdleMatchWakeup Idle Match Wakeup.

kLpuartMatchOnAndMatchOff Match On and Match Off.

kLpuartEnablesRwuOnDataMatch Enables RWU on Data Match and Match On/Off for transmitter CTS input.

20.2.3.11 enum lpuart_ir_tx_pulsewidth_t

Enumerator

kLpuartIrThreeSixteenthsWidth 3/16 pulse

kLpuartIrOneSixteenthWidth 1/16 pulse

kLpuartIrOneThirtysecondsWidth 1/32 pulse

kLpuartIrOneFourthWidth 1/4 pulse

20.2.3.12 enum lpuart_idle_char_t

Enumerator

kLpuart_1_IdleChar 1 idle character

kLpuart_2_IdleChar 2 idle character

kLpuart_4_IdleChar 4 idle character

kLpuart_8_IdleChar 8 idle character

kLpuart_16_IdleChar 16 idle character

kLpuart_32_IdleChar 32 idle character

kLpuart_64_IdleChar 64 idle character

kLpuart_128_IdleChar 128 idle character

20.2.3.13 enum lpuart_cts_source_t

Configures the source of the CTS input.

Enumerator

kLpuartCtsSourcePin CTS input is the LPUART_CTS pin.

kLpuartCtsSourceInvertedReceiverMatch CTS input is the inverted Receiver Match result.

20.2.3.14 enum lpuart_cts_config_t

Enumerator

kLpuartCtsSampledOnEachChar CTS input is sampled at the start of each character.

kLpuartCtsSampledOnIdle CTS input is sampled when the transmitter is idle.

20.2.3.15 enum lpuart_status_flag_t

This provides constants for the LPUART status flags for use in the UART functions.

Enumerator

kLpuartTxDataRegEmpty Tx data register empty flag, sets when Tx buffer is empty.

kLpuartTxComplete Transmission complete flag, sets when transmission activity complete.

kLpuartRxDataRegFull Rx data register full flag, sets when the receive data buffer is full.

kLpuartIdleLineDetect Idle line detect flag, sets when idle line detected.

kLpuartRxOverrun Rxr Overrun, sets when new data is received before data is read from receive register.

kLpuartNoiseDetect Rxr takes 3 samples of each received bit. If any of these samples differ, noise flag sets

kLpuartFrameErr Frame error flag, sets if logic 0 was detected where stop bit expected.

kLpuartParityErr If parity enabled, sets upon parity error detection.

kLpuartLineBreakDetect LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.

kLpuartRxActiveEdgeDetect Rx pin active edge interrupt flag, sets when active edge detected.

kLpuartRxActive Receiver Active Flag (RAF), sets at beginning of valid start bit.

kLpuartNoiseInCurrentWord NOISY bit, sets if noise detected in current data word.

kLpuartParityErrInCurrentWord PARITYE bit, sets if noise detected in current data word.

20.2.3.16 enum lpuart_interrupt_t

Enumerator

kLpuartIntLinBreakDetect LIN break detect.

kLpuartIntRxActiveEdge RX Active Edge.

kLpuartIntTxDataRegEmpty Transmit data register empty.

kLpuartIntTxComplete Transmission complete.

kLpuartIntRxDataRegFull Receiver data register full.

kLpuartIntIdleLine Idle line.

kLpuartIntRxOverrun Receiver Overrun.

kLpuartIntNoiseErrFlag Noise error flag.

kLpuartIntFrameErrFlag Framing error flag.

kLpuartIntParityErrFlag Parity error flag.

20.2.4 Function Documentation

20.2.4.1 void LPUART_HAL_Init(uint32_t *baseAddr*)

Parameters

<i>baseAddr</i>	LPUART base address.
-----------------	----------------------

20.2.4.2 static void LPUART_HAL_SetTransmitterCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address.
<i>enable</i>	Enable(true) or disable(false) transmitter.

20.2.4.3 static bool LPUART_HAL_GetTransmitterCmd (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

State of LPUART transmitter enable(true)/disable(false)

20.2.4.4 static void LPUART_HAL_SetReceiverCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	Enable(true) or disable(false) receiver.

20.2.4.5 static bool LPUART_HAL_GetReceiverCmd (*uint32_t baseAddr*) [inline], [static]

LPUART HAL driver

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

State of LPUART receiver enable(true)/disable(false)

20.2.4.6 **Ipuart_status_t LPUART_HAL_SetBaudRate (uint32_t *baseAddr*, uint32_t *sourceClockInHz*, uint32_t *desiredBaudRate*)**

In some LPUART instances the user must disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address.
<i>sourceClockInHz</i>	LPUART source input clock in Hz.
<i>desiredBaudRate</i>	LPUART desired baud rate.

Returns

An error code or kStatus_Success

20.2.4.7 **static void LPUART_HAL_SetBaudRateDivisor (uint32_t *baseAddr*, uint32_t *baudRateDivisor*) [inline], [static]**

Parameters

<i>baseAddr</i>	LPUART base address.
<i>baudRateDivisor</i>	The baud rate modulo division "SBR"

20.2.4.8 **void LPUART_HAL_SetBitCountPerChar (uint32_t *baseAddr*, Ipuart_bit_count_per_char_t *bitCountPerChar*)**

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address.
<i>bitCountPerChar</i>	Number of bits per char (8, 9, or 10, depending on the LPUART instance)

20.2.4.9 void LPUART_HAL_SetParityMode (uint32_t *baseAddr*, lpuart_parity_mode_t *parityModeType*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address.
<i>parityModeType</i>	Parity mode (enabled, disable, odd, even - see parity_mode_t struct)

20.2.4.10 static void LPUART_HAL_SetStopBitCount (uint32_t *baseAddr*, lpuart_stop_bit_count_t *stopBitCount*) [inline], [static]

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address.
<i>stopBitCount</i>	Number of stop bits (1 or 2 - see lpuart_stop_bit_count_t struct)

Returns

An error code (an unsupported setting in some LPUARTs) or kStatus_Success

20.2.4.11 static void LPUART_HAL_SetTxInversionCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function should only be called when the LPUART is between transmit and receive packets.

LPUART HAL driver

Parameters

<i>baseAddr</i>	LPUART base address.
<i>enable</i>	Enable (1) or disable (0) transmit inversion

20.2.4.12 static void LPUART_HAL_SetRxInversionCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

This function should only be called when the LPUART is between transmit and receive packets.

Parameters

<i>baseAddr</i>	LPUART base address.
<i>enable</i>	Enable (1) or disable (0) receive inversion

20.2.4.13 static uint32_t LPUART_HAL_GetDataRegAddr (*uint32_t baseAddr*) [inline], [static]

Returns

LPUART tx/rx data register address.

20.2.4.14 void LPUART_HAL_SetIntMode (*uint32_t baseAddr, lpuart_interrupt_t interrupt, bool enable*)

Parameters

<i>baseAddr</i>	LPUART module base address.
<i>interrupt</i>	LPUART interrupt configuration data.
<i>enable</i>	true: enable, false: disable.

20.2.4.15 bool LPUART_HAL.GetIntMode (*uint32_t baseAddr, lpuart_interrupt_t interrupt*)

Parameters

<i>baseAddr</i>	LPUART module base address.
<i>interrupt</i>	LPUART interrupt configuration data.

Returns

true: enable, false: disable.

20.2.4.16 static void LPUART_HAL_SetTxDataRegEmptyIntCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	true: enable, false: disable.

20.2.4.17 static bool LPUART_HAL_GetTxDataRegEmptyIntCmd (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

Bit setting of the interrupt enable bit

20.2.4.18 static void LPUART_HAL_SetRxDataRegFullIntCmd (*uint32_t baseAddr, bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

LPUART HAL driver

<i>enable</i>	true: enable, false: disable.
---------------	-------------------------------

**20.2.4.19 static bool LPUART_HAL_GetRxDataRegFullIntCmd (uint32_t *baseAddr*)
[inline], [static]**

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

Bit setting of the interrupt enable bit

**20.2.4.20 static void LPUART_HAL_Putchar (uint32_t *baseAddr*, uint8_t *data*)
[inline], [static]**

Parameters

<i>baseAddr</i>	LPUART Instance
<i>data</i>	data to send (8-bit)

20.2.4.21 void LPUART_HAL_Putchar9 (uint32_t *baseAddr*, uint16_t *data*)

Parameters

<i>baseAddr</i>	LPUART Instance
<i>data</i>	data to send (9-bit)

20.2.4.22 lpuart_status_t LPUART_HAL_Putchar10 (uint32_t *baseAddr*, uint16_t *data*)

Parameters

<i>baseAddr</i>	LPUART Instance
<i>data</i>	data to send (10-bit)

Returns

An error code or kStatus_Success

20.2.4.23 **static void LPUART_HAL_Getchar (uint32_t *baseAddr*, uint8_t * *readData*) [inline], [static]**

LPUART HAL driver

Parameters

<i>baseAddr</i>	LPUART base address
<i>readData</i>	Data read from receive (8-bit)

20.2.4.24 void LPUART_HAL_Getchar9 (uint32_t *baseAddr*, uint16_t * *readData*)

Parameters

<i>baseAddr</i>	LPUART base address
<i>readData</i>	Data read from receive (9-bit)

20.2.4.25 void LPUART_HAL_Getchar10 (uint32_t *baseAddr*, uint16_t * *readData*)

Parameters

<i>baseAddr</i>	LPUART base address
<i>readData</i>	Data read from receive (10-bit)

20.2.4.26 void LPUART_HAL_SendDataPolling (uint32_t *baseAddr*, const uint8_t * *txBuff*, uint32_t *txSize*)

This function only supports 8-bit transaction.

Parameters

<i>baseAddr</i>	LPUART module base address.
<i>txBuff</i>	The buffer pointer which saves the data to be sent.
<i>txSize</i>	Size of data to be sent in unit of byte.

20.2.4.27 Ipuart_status_t LPUART_HAL_ReceiveDataPolling (uint32_t *baseAddr*, uint8_t * *rxBuff*, uint32_t *rxSize*)

This function only supports 8-bit transaction.

Parameters

<i>baseAddr</i>	LPUART module base address.
<i>rxBuff</i>	The buffer pointer which saves the data to be received.
<i>rxSize</i>	Size of data need to be received in unit of byte.

Returns

Whether the transaction is success or rx overrun.

20.2.4.28 static void LPUART_HAL_SetIdleChar (uint32_t *baseAddr*, lpuart_idle_char_t *idleConfig*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>idleConfig</i>	Idle characters configuration

20.2.4.29 static lpuart_idle_char_t LPUART_HAL_GetIdleChar (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

idle characters configuration

20.2.4.30 static bool LPUART_HAL_IsCurrentDataWithNoise (uint32_t *baseAddr*) [inline], [static]

Parameters

LPUART HAL driver

<i>baseAddr</i>	LPUART base address.
-----------------	----------------------

Returns

The status of the NOISY bit in the LPUART extended data register

20.2.4.31 static bool LPUART_HAL_IsCurrentDataWithFrameError (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

The status of the FRETSC bit in the LPUART extended data register

20.2.4.32 static void LPUART_HAL_SetTxSpecialChar (uint32_t *baseAddr*, uint8_t *specialChar*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>specialChar</i>	T9 is used to indicate a break character when 0 an idle character when 1, the contents of DATA[T8:T0] should be zero.

20.2.4.33 static bool LPUART_HAL_IsCurrentDataWithParityError (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

The status of the PARITYE bit in the LPUART extended data register

20.2.4.34 static bool LPUART_HAL_IsReceiveBufferEmpty (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

TRUE if the receive-buffer is empty, else FALSE.

20.2.4.35 static bool LPUART_HAL_WasPreviousReceiverStatIdle (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

TRUE if the previous BUS state was IDLE, else FALSE.

20.2.4.36 static void LPUART_HAL_SetWaitModeOperation (uint32_t *baseAddr*, lpuart_operation_config_t *mode*) [inline], [static]

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address
<i>mode</i>	LPUART wait mode operation - operates or stops to operate in wait mode.

20.2.4.37 static lpuart_operation_config_t LPUART_HAL_GetWaitModeOperation (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

LPUART wait mode operation configuration

- kLpuartOperates or KLpuartStops in wait mode

LPUART HAL driver

20.2.4.38 void LPUART_HAL_SetLoopbackCmd (uint32_t *baseAddr*, bool *enable*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	LPUART loopback mode - disabled (0) or enabled (1)

20.2.4.39 void LPUART_HAL_SetSingleWireCmd (uint32_t *baseAddr*, bool *enable*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	LPUART loopback mode - disabled (0) or enabled (1)

20.2.4.40 static void LPUART_HAL_SetTxdirInSinglewireMode (uint32_t *baseAddr*, lpuart_singlewire_txdir_t *direction*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>direction</i>	LPUART single-wire transmit direction - input or output

20.2.4.41 lpuart_status_t LPUART_HAL_SetReceiverInStandbyMode (uint32_t *baseAddr*)

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

Error code or kStatus_Success

20.2.4.42 static void LPUART_HAL_PutReceiverInNormalMode (uint32_t *baseAddr*) [inline], [static]

LPUART HAL driver

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

20.2.4.43 static bool LPUART_HAL_IsReceiverInStandby (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

LPUART in normal mode (0) or standby (1)

20.2.4.44 static void LPUART_HAL_SetReceiverWakeupMode (uint32_t *baseAddr*, lpuart_wakeup_method_t *method*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>method</i>	LPUART wakeup method: 0 - Idle-line wake (default), 1 - addr-mark wake

20.2.4.45 static lpuart_wakeup_method_t LPUART_HAL_GetReceiverWakeupMode (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

LPUART wakeup method: kLpuartIdleLineWake: 0 - Idle-line wake (default), kLpuartAddrMarkWake: 1 - addr-mark wake

20.2.4.46 void LPUART_HAL_SetIdleLineDetect (uint32_t *baseAddr*, const lpuart_idle_line_config_t * *config*)

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address
<i>config</i>	LPUART configuration data for idle line detect operation

20.2.4.47 static void LPUART_HAL_SetBreakCharTransmitLength (*uint32_t baseAddr*, *lpuart_break_char_length_t length*) [inline], [static]

In some LPUART instances, the user should disable the transmitter before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address
<i>length</i>	LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times

20.2.4.48 static void LPUART_HAL_SetBreakCharDetectLength (*uint32_t baseAddr*, *lpuart_break_char_length_t length*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>length</i>	LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times

20.2.4.49 static void LPUART_HAL_QueueBreakCharToSend (*uint32_t baseAddr*, *bool enable*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	LPUART normal/queue break char - disabled (normal mode, default: 0) or enabled (queue break char: 1)

20.2.4.50 static void LPUART_HAL_SetMatchAddressMode (*uint32_t baseAddr*, *lpuart_match_config_t config*) [inline], [static]

LPUART HAL driver

Parameters

<i>baseAddr</i>	LPUART base address
<i>config</i>	MATCFG: Configures the match addressing mode used.

20.2.4.51 void LPUART_HAL_SetMatchAddressReg1 (uint32_t *baseAddr*, bool *enable*, uint8_t *value*)

The MAEN bit must be cleared before configuring MA value, so the enable/disable and set value must be included inside one function.

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	Match address model enable (true)/disable (false)
<i>value</i>	Match address value to program into match address register 1

20.2.4.52 void LPUART_HAL_SetMatchAddressReg2 (uint32_t *baseAddr*, bool *enable*, uint8_t *value*)

The MAEN bit must be cleared before configuring MA value, so the enable/disable and set value must be included inside one function.

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	Match address model enable (true)/disable (false)
<i>value</i>	Match address value to program into match address register 2

20.2.4.53 static void LPUART_HAL_SetSendMsbFirstCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

In some LPUART instances, the user should disable the transmitter/receiver before calling this function. Generally, this may be applied to all LPUARTs to ensure safe operation.

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	false - LSB (default, disabled), true - MSB (enabled)

20.2.4.54 static void LPUART_HAL_SetReceiveResyncCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
<i>enable</i>	re-sync of received data word configuration: true - re-sync of received data word (default) false - disable the re-sync

20.2.4.55 bool LPUART_HAL_GetStatusFlag (uint32_t *baseAddr*, lpuart_status_flag_t *statusFlag*)

Parameters

<i>baseAddr</i>	LPUART base address
<i>statusFlag</i>	The status flag to query

Returns

Whether the current status flag is set(true) or not(false).

20.2.4.56 static bool LPUART_HAL_IsTxDataRegEmpty (uint32_t *baseAddr*) [inline], [static]

This function returns the state of the LPUART Transmit data register empty flag.

Parameters

<i>baseAddr</i>	LPUART module base address.
-----------------	-----------------------------

Returns

The status of Transmit data register empty flag, which is set when transmit buffer is empty.

20.2.4.57 static bool LPUART_HAL_IsRxDataRegFull (uint32_t *baseAddr*) [inline], [static]

LPUART HAL driver

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

Status of the receive data register full flag, sets when the receive data buffer is full.

20.2.4.58 static bool LPUART_HAL_IsTxComplete (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	LPUART base address
-----------------	---------------------

Returns

Status of Transmission complete flag, sets when transmitter is idle (transmission activity complete)

20.2.4.59 lpuart_status_t LPUART_HAL_ClearStatusFlag (uint32_t *baseAddr*, lpuart_status_flag_t *statusFlag*)

Parameters

<i>baseAddr</i>	LPUART base address
<i>statusFlag</i>	Desired LPUART status flag to clear

Returns

An error code or kStatus_Success

20.3 LPUART Peripheral driver

20.3.1 Overview

This section describes the programming interface of the LPUART Peripheral driver.

Data Structures

- struct [lpuart_state_t](#)
Runtime state of the LPUART driver. [More...](#)
- struct [lpuart_user_config_t](#)
LPUART configuration structure. [More...](#)
- struct [lpuart_edma_state_t](#)
Runtime state structure for UART driver with DMA. [More...](#)
- struct [lpuart_edma_user_config_t](#)
LPUART configuration structure. [More...](#)

Typedefs

- [typedef void\(* lpuart_rx_callback_t \)](#)(uint32_t instance, void *lpuartState)
LPUART receive callback function type.

Variables

- const uint32_t [g_lpuartBaseAddr](#) [HW_LPUART_INSTANCE_COUNT]
Table of base addresses for LPUART instances.
- const IRQn_Type [g_lpuartRxTxIrqId](#) [HW_LPUART_INSTANCE_COUNT]
Table to save LPUART IRQ enumeration numbers defined in the CMSIS header file.
- const uint32_t [g_lpuartBaseAddr](#) [HW_LPUART_INSTANCE_COUNT]
Table of base addresses for LPUART instances.

LPUART Driver

- [lpuart_status_t LPUART_DRV_Init](#) (uint32_t instance, [lpuart_state_t](#) *lpuartStatePtr, const [lpuart_user_config_t](#) *lpuartUserConfig)
Initializes an LPUART operation instance.
- [void LPUART_DRV_Deinit](#) (uint32_t instance)
Shuts down the LPUART by disabling interrupts and transmitter/receiver.
- [lpuart_rx_callback_t LPUART_DRV_InstallRxCallback](#) (uint32_t instance, [lpuart_rx_callback_t](#) function, uint8_t *rxBuff, void *callbackParam, bool alwaysEnableRxIrq)
Installs callback function for the LPUART receive.
- [lpuart_status_t LPUART_DRV_SendDataBlocking](#) (uint32_t instance, const uint8_t *txBuff, uint32_t txSize, uint32_t timeout)
Sends data out through the LPUART module using a blocking method.

LPUART Peripheral driver

- `lpuart_status_t LPUART_DRV_SendData` (uint32_t instance, const uint8_t *txBuff, uint32_t txSize)
Sends data out through the LPUART module using a non-blocking method.
- `lpuart_status_t LPUART_DRV_GetTransmitStatus` (uint32_t instance, uint32_t *bytesRemaining)
Returns whether the previous transmit is complete.
- `lpuart_status_t LPUART_DRV_AbortSendingData` (uint32_t instance)
Terminates a non-blocking transmission early.
- `lpuart_status_t LPUART_DRV_ReceiveDataBlocking` (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize, uint32_t timeout)
Gets data from the LPUART module by using a blocking method.
- `lpuart_status_t LPUART_DRV_ReceiveData` (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize)
Gets data from the LPUART module by using a non-blocking method.
- `lpuart_status_t LPUART_DRV_GetReceiveStatus` (uint32_t instance, uint32_t *bytesRemaining)
Returns whether the previous receive is complete.
- `lpuart_status_t LPUART_DRV_AbortReceivingData` (uint32_t instance)
Terminates a non-blocking receive early.

LPUART DMA Driver

- `lpuart_status_t LPUART_DRV_EdmaInit` (uint32_t instance, `lpuart_edma_state_t` *lpuartEdmaStatePtr, const `lpuart_edma_user_config_t` *lpuartUserConfig)
Initializes an LPUART instance to work with DMA.
- `void LPUART_DRV_EdmaDeinit` (uint32_t instance)
Shuts down the LPUART.
- `lpuart_status_t LPUART_DRV_EdmaSendDataBlocking` (uint32_t instance, const uint8_t *txBuff, uint32_t txSize, uint32_t timeout)
Sends (transmits) data out through the LPUART-DMA module using a blocking method.
- `lpuart_status_t LPUART_DRV_EdmaSendData` (uint32_t instance, const uint8_t *txBuff, uint32_t txSize)
Sends (transmits) data through the LPUART-DMA module using a non-blocking method.
- `lpuart_status_t LPUART_DRV_EdmaGetTransmitStatus` (uint32_t instance, uint32_t *bytesRemaining)
Returns whether the previous LPUART-DMA transmit has finished.
- `lpuart_status_t LPUART_DRV_EdmaAbortSendingData` (uint32_t instance)
Terminates a non-blocking LPUART-DMA transmission early.
- `lpuart_status_t LPUART_DRV_EdmaReceiveDataBlocking` (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize, uint32_t timeout)
Gets (receives) data from the LPUART-DMA module using a blocking method.
- `lpuart_status_t LPUART_DRV_EdmaReceiveData` (uint32_t instance, uint8_t *rxBuff, uint32_t rxSize)
Gets (receives) data from the LPUART-DMA module using a non-blocking method.
- `lpuart_status_t LPUART_DRV_EdmaGetReceiveStatus` (uint32_t instance, uint32_t *bytesRemaining)
Returns whether the previous LPUART-DMA receive is complete.
- `lpuart_status_t LPUART_DRV_EdmaAbortReceivingData` (uint32_t instance)
Terminates a non-blocking LPUART-DMA receive early.

20.3.2 Data Structure Documentation

20.3.2.1 struct lpuart_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.
- volatile size_t **txSize**
The remaining number of bytes to be transmitted.
- volatile size_t **rxSize**
The remaining number of bytes to be received.
- volatile bool **isTxBusy**
True if there is an active transmit.
- volatile bool **isRxBusy**
True if there is an active receive.
- volatile bool **isTxBlocking**
True if transmit is blocking transaction.
- volatile bool **isRxBlocking**
True if receive is blocking transaction.
- semaphore_t **txIrqSync**
Used to wait for ISR to complete its Tx business.
- semaphore_t **rxIrqSync**
Used to wait for ISR to complete its Rx business.
- lpuart_rx_callback_t **rxCallback**
Callback to invoke after receiving byte.
- void * **rxCallbackParam**
Receive callback parameter pointer.

LPUART Peripheral driver

20.3.2.1.0.43 Field Documentation

20.3.2.1.0.43.1 `const uint8_t* lpuart_state_t::txBuff`

20.3.2.1.0.43.2 `uint8_t* lpuart_state_t::rxBuff`

20.3.2.1.0.43.3 `volatile size_t lpuart_state_t::txSize`

20.3.2.1.0.43.4 `volatile size_t lpuart_state_t::rxSize`

20.3.2.1.0.43.5 `volatile bool lpuart_state_t::isTxBusy`

20.3.2.1.0.43.6 `volatile bool lpuart_state_t::isRxBusy`

20.3.2.1.0.43.7 `volatile bool lpuart_state_t::isTxBlocking`

20.3.2.1.0.43.8 `volatile bool lpuart_state_t::isRxBlocking`

20.3.2.1.0.43.9 `semaphore_t lpuart_state_t::txIrqSync`

20.3.2.1.0.43.10 `semaphore_t lpuart_state_t::rxIrqSync`

20.3.2.1.0.43.11 `lpuart_rx_callback_t lpuart_state_t::rxCallback`

20.3.2.1.0.43.12 `void* lpuart_state_t::rxCallbackParam`

20.3.2.2 struct lpuart_user_config_t

Data Fields

- `clock_lpuart_src_t clockSource`
LPUART clock source in fsl_sim_hal_<device>.h.
- `uint32_t baudRate`
LPUART baud rate.
- `lpuart_parity_mode_t parityMode`
parity mode, disabled (default), even, odd
- `lpuart_stop_bit_count_t stopBitCount`
number of stop bits, 1 stop bit (default) or 2 stop bits
- `lpuart_bit_count_per_char_t bitCountPerChar`
number of bits, 8-bit (default) or 9-bit in a char (up to 10-bits in some LPUART instances).

20.3.2.2.0.44 Field Documentation

20.3.2.2.0.44.1 `lpuart_bit_count_per_char_t lpuart_user_config_t::bitCountPerChar`

20.3.2.3 struct lpuart_edma_state_t

Data Fields

- `volatile bool isTxBusy`
True if there is an active transmit.

- volatile bool **isRxBusy**
True if there is an active receive.
- volatile bool **isTxBlocking**
True if transmit is blocking transaction.
- volatile bool **isRxBlocking**
True if receive is blocking transaction.
- **semaphore_t txIrqSync**
Used to wait for ISR to complete its TX business.
- **semaphore_t rxIrqSync**
Used to wait for ISR to complete its RX business.
- **edma_chn_state_t edmaLpuartTx**
Structure definition for the eDMA channel.
- **edma_chn_state_t edmaLpuartRx**
Structure definition for the eDMA channel.

20.3.2.3.0.45 Field Documentation

20.3.2.3.0.45.1 volatile bool lpuart_edma_state_t::isTxBusy

20.3.2.3.0.45.2 volatile bool lpuart_edma_state_t::isRxBusy

20.3.2.3.0.45.3 volatile bool lpuart_edma_state_t::isTxBlocking

20.3.2.3.0.45.4 volatile bool lpuart_edma_state_t::isRxBlocking

20.3.2.3.0.45.5 semaphore_t lpuart_edma_state_t::txIrqSync

20.3.2.3.0.45.6 semaphore_t lpuart_edma_state_t::rxIrqSync

20.3.2.4 struct lpuart_edma_user_config_t

Data Fields

- **clock_lpuart_src_t clockSource**
LPUART clock source in `fsl_sim_hal_<device>.h`.
- **uint32_t baudRate**
LPUART baud rate.
- **lpuart_parity_mode_t parityMode**
parity mode, disabled (default), even, odd
- **lpuart_stop_bit_count_t stopBitCount**
number of stop bits, 1 stop bit (default) or 2 stop bits
- **lpuart_bit_count_per_char_t bitCountPerChar**
number of bits, 8-bit (default) or 9-bit in a char (up to 10-bits in some LPUART instances).

LPUART Peripheral driver

20.3.2.4.0.46 Field Documentation

20.3.2.4.0.46.1 `lpuart_bit_count_per_char_t lpuart_edma_user_config_t::bitCountPerChar`

20.3.3 Typedef Documentation

20.3.3.1 `typedef void(* lpuart_rx_callback_t)(uint32_t instance, void *lpuartState)`

20.3.4 Function Documentation

20.3.4.1 `lpuart_status_t LPUART_DRV_Init (uint32_t instance, lpuart_state_t * lpuartStatePtr, const lpuart_user_config_t * lpuartUserConfig)`

The caller provides memory for the driver state structures during initialization. The user must select the LPUART clock source in the application to initialize the LPUART.

Parameters

<i>instance</i>	LPUART instance number
<i>lpuartUserConfig</i>	user configuration structure of type lpuart_user_config_t
<i>lpuartStatePtr</i>	pointer to the LPUART driver state structure

Returns

An error code or kStatus_LPUART_Success

20.3.4.2 void LPUART_DRV_Deinit (uint32_t *instance*)

Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

20.3.4.3 lpuart_rx_callback_t LPUART_DRV_InstallRxCallback (uint32_t *instance*, lpuart_rx_callback_t *function*, uint8_t * *rxBuff*, void * *callbackParam*, bool *alwaysEnableRxIrq*)

Parameters

<i>instance</i>	The LPUART instance number.
<i>function</i>	The LPUART receive callback function.
<i>rxBuff</i>	The receive buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive.
<i>callbackParam</i>	The LPUART receive callback parameter pointer.
<i>alwaysEnableRxIrq</i>	Whether always enable Rx IRQ or not.

Returns

Former LPUART receive callback function pointer.

20.3.4.4 lpuart_status_t LPUART_DRV_SendDataBlocking (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*, uint32_t *timeout*)

Blocking means that the function does not return until the transmission is complete.

LPUART Peripheral driver

Parameters

<i>instance</i>	LPUART instance number
<i>txBuff</i>	source buffer containing 8-bit data chars to send
<i>txSize</i>	the number of bytes to send
<i>timeout</i>	timeout value for RTOS abstraction sync control

Returns

An error code or kStatus_LPUART_Success

20.3.4.5 lpuart_status_t LPUART_DRV_SendData (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*)

This enables an async method for transmitting data. When used with a non-blocking receive, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete.

Parameters

<i>instance</i>	LPUART instance number
<i>txBuff</i>	source buffer containing 8-bit data chars to send
<i>txSize</i>	the number of bytes to send

Returns

An error code or kStatus_LPUART_Success

20.3.4.6 lpuart_status_t LPUART_DRV_GetTransmitStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)

Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

<i>bytesRemaining</i>	Pointer to value that is populated with the number of bytes that have been sent in the active transfer
-----------------------	--

Returns

The transmit status.

Return values

<i>kStatus_LPUART_Success</i>	The transmit has completed successfully.
<i>kStatus_LPUART_TxBusy</i>	The transmit is still in progress. <i>bytesTransmitted</i> will be filled with the number of bytes that have been transmitted so far.

20.3.4.7 **lpuart_status_t LPUART_DRV_AbortSendingData (uint32_t instance)**

Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

Returns

Whether the aborting is successful or not.

20.3.4.8 **lpuart_status_t LPUART_DRV_ReceiveDataBlocking (uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout)**

Blocking means that the function does not return until the receive is complete.

Parameters

<i>instance</i>	LPUART instance number
<i>rxBuff</i>	buffer containing 8-bit read data chars received
<i>rxSize</i>	the number of bytes to receive
<i>timeout</i>	timeout value for RTOS abstraction sync control

Returns

An error code or *kStatus_LPUART_Success*

LPUART Peripheral driver

20.3.4.9 **lpuart_status_t LPUART_DRV_ReceiveData (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*)**

This enables an async method for receiving data. When used with a non-blocking transmission, the LPUART can perform a full duplex operation. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the receive is complete.

Parameters

<i>instance</i>	LPUART instance number
<i>rxBuff</i>	buffer containing 8-bit read data chars received
<i>rxSize</i>	the number of bytes to receive

Returns

An error code or kStatus_LPUART_Success

20.3.4.10 **lpuart_status_t LPUART_DRV_GetReceiveStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

Parameters

<i>instance</i>	LPUART instance number
<i>bytesRemaining</i>	pointer to value that is filled with the number of bytes that still need to be received in the active transfer.

Returns

The receive status.

Return values

<i>kStatus_LPUART_Success</i>	the receive has completed successfully.
<i>kStatus_LPUART_Rx-Busy</i>	the receive is still in progress. <i>bytesReceived</i> will be filled with the number of bytes that have been received so far.

20.3.4.11 **lpuart_status_t LPUART_DRV_AbortReceivingData (uint32_t *instance*)**

Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

Returns

Whether the receiving was successful or not.

20.3.4.12 **lpuart_status_t LPUART_DRV_EdmalInit (uint32_t *instance*, lpuart_edma_state_t * *lpuartEdmaStatePtr*, const lpuart_edma_user_config_t * *lpuartUserConfig*)**

This function initializes the run-time state structure to keep track of the on-going transfers, ungates the clock to the LPUART module, initializes the module to user-defined settings and default settings, configures the IRQ state structure and enables the module-level interrupt to the core, and enables the LPUART module transmitter and receiver. This example shows how to set up the [lpuart_edma_state_t](#) and the [lpuart_user_config_t](#) parameters and how to call the LPUART_DRV_EdmalInit function by passing in these parameters:

```
lpuart_user_config_t lpuartConfig;
lpuartConfig.baudRate = 9600;
lpuartConfig.bitCountPerChar = kLpuart8BitsPerChar;
lpuartConfig.parityMode = kLpuartParityDisabled;
lpuartConfig.stopBitCount = kLpuartOneStopBit;
lpuart_edma_state_t lpuartEdmaState;
LPUART_DRV_EdmalInit(instance, &lpuartEdmaState, &lpuartConfig);
```

Parameters

<i>instance</i>	The LPUART instance number.
<i>lpuartEdma-StatePtr</i>	A pointer to the LPUART driver state structure memory. The user passes in the memory for the run-time state structure. The LPUART driver populates the members. This run-time state structure keeps track of the current transfer in progress.
<i>lpuartUser-Config</i>	The user configuration structure of type lpuart_user_config_t . The user populates the members of this structure and passes the pointer of this structure into this function.

Returns

An error code or kStatus_LPUART_Success.

20.3.4.13 **void LPUART_DRV_EdmaDeinit (uint32_t *instance*)**

This function disables the LPUART-DMA trigger, the transmitter, and the receiver.

LPUART Peripheral driver

Parameters

<i>instance</i>	The LPUART instance number.
-----------------	-----------------------------

20.3.4.14 **lpuart_status_t LPUART_DRV_EdmaSendDataBlocking (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*, uint32_t *timeout*)**

Parameters

<i>instance</i>	The LPUART instance number.
<i>txBuff</i>	A pointer to the source buffer containing 8-bit data chars to send.
<i>txSize</i>	The number of bytes to send.
<i>timeout</i>	A timeout value for RTOS abstraction sync control in milliseconds (ms).

Returns

An error code or kStatus_LPUART_Success.

20.3.4.15 **lpuart_status_t LPUART_DRV_EdmaSendData (uint32_t *instance*, const uint8_t * *txBuff*, uint32_t *txSize*)**

Parameters

<i>instance</i>	The LPUART module base address.
<i>txBuff</i>	A pointer to the source buffer containing 8-bit data chars to send.
<i>txSize</i>	The number of bytes to send.

Returns

An error code or kStatus_LPUART_Success.

20.3.4.16 **lpuart_status_t LPUART_DRV_EdmaGetTransmitStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

Parameters

<i>instance</i>	The LPUART module base address.
<i>bytesRemaining</i>	A pointer to a value that is populated with the number of bytes that are remaining in the active transfer.

Returns

Current transmit status.

Return values

<i>kStatus_LPUART_Success</i>	The transmit has completed successfully.
<i>kStatus_LPUART_TxBusy</i>	The transmit is still in progress. <i>bytesTransmitted</i> is filled with the number of bytes which are transmitted up to that point.

20.3.4.17 **lpuart_status_t LPUART_DRV_EdmaAbortSendingData (uint32_t *instance*)**

Parameters

<i>instance</i>	The LPUART module base address.
-----------------	---------------------------------

Returns

Whether the abort of transmitting was successful or not.

Return values

<i>kStatus_LPUART_Success</i>	The transmit was successful.
<i>kStatus_LPUART_No_TransmitInProgress</i>	No transmission is currently in progress.

20.3.4.18 **lpuart_status_t LPUART_DRV_EdmaReceiveDataBlocking (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*, uint32_t *timeout*)**

LPUART Peripheral driver

Parameters

<i>instance</i>	The LPUART module base address.
<i>rxBuff</i>	A pointer to the buffer containing 8-bit read data chars received.
<i>rxSize</i>	The number of bytes to receive.
<i>timeout</i>	A timeout value for RTOS abstraction sync control in milliseconds (ms).

Returns

An error code or kStatus_LPUART_Success.

20.3.4.19 **lpuart_status_t LPUART_DRV_EdmaReceiveData (uint32_t *instance*, uint8_t * *rxBuff*, uint32_t *rxSize*)**

Parameters

<i>instance</i>	The LPUART module base address.
<i>rxBuff</i>	A pointer to the buffer containing 8-bit read data chars received.
<i>rxSize</i>	The number of bytes to receive.

Returns

An error code or kStatus_LPUART_Success.

20.3.4.20 **lpuart_status_t LPUART_DRV_EdmaGetReceiveStatus (uint32_t *instance*, uint32_t * *bytesRemaining*)**

Parameters

<i>instance</i>	The LPUART module base address.
<i>bytes-Remaining</i>	A pointer to a value that populated with the number of bytes which still need to be received in the active transfer.

Returns

Current receiving status.

Return values

<i>kStatus_LPUART_Success</i>	The receive has completed successfully.
<i>kStatus_LPUART_RxBusy</i>	The receive is still in progress. <i>bytesReceived</i> is filled with the number of bytes which are received up to that point.

20.3.4.21 **Ipuart_status_t LPUART_DRV_EdmaAbortReceivingData (uint32_t instance)**

Parameters

<i>instance</i>	The LPUART module base address.
-----------------	---------------------------------

Returns

Whether the abort of receiving was successful or not.

Return values

<i>kStatus_LPUART_Success</i>	The receive was successful.
<i>kStatus_LPUART_NoTransmitInProgress</i>	No receive is currently in progress.

20.3.5 Variable Documentation

20.3.5.1 const uint32_t g_IpuartBaseAddr[HW_LPUART_INSTANCE_COUNT]

20.3.5.2 const uint32_t g_IpuartBaseAddr[HW_LPUART_INSTANCE_COUNT]

LPUART Type Definitions

20.4 LPUART Type Definitions

This section describes the LPUART type definitions.

20.4.1 LPUART Overview

The LPUART peripheral driver transfers data to and from external devices on the Low Power Universal Asynchronous Receiver/Transmitter (LPUART) serial bus. It provides a way to transmit or receive buffers of data with a single function calls.

20.4.2 LPUART Device structures

The driver uses instantiations of the lpuart_tx_state_t and the lpuart_rx_state_t structure to maintain the current state of a particular LPUART instance module driver. The user is required to provide memory for the driver state structures during the initialization. The driver itself does not statically allocate memory. The structures are provided below:

```
// Runtime transmit state of the LPUART driver.
typedef struct LpuartTxState {
    uint32_t instance;
    bool isTransmitInProgress;
    bool isTransmitAsync;
    const uint8_t * sendBuffer;
    size_t remainingSendByteCount;
    size_t transmittedByteCount;
    sync_object_t irqSync;
    uint8_t fifoEntryCount;
} lpuart_tx_state_t;

// Runtime receive state of the LPUART driver.
typedef struct LpuartRxState {
    uint32_t instance;
    bool isReceiveInProgress;
    bool isReceiveAsync;
    uint8_t * receiveBuffer;
    size_t remainingReceiveByteCount;
    size_t receivedByteCount;
    sync_object_t irqSync;
    uint8_t fifoEntryCount;
} lpuart_rx_state_t;
```

20.4.3 LPUART Initialization

1. To initialize the LPUART driver, call the [LPUART_DRV_Init\(\)](#) function and pass the instance number of the LPUART peripheral you want to use. For example, to use LPUART0 pass a value 0 to the initialization function.
2. Pass a user configuration structure [lpuart_user_config_t](#) as shown here:

```
// LPUART configuration structure
typedef struct LpuartUserConfig {
    uint32_t baudRate;
```

```

lpuart_parity_mode_t parityMode;
lpuart_stop_bit_count_t stopBitCount;
lpuart_bit_count_per_char_t bitCountPerChar;
} lpuart_user_config_t;

```

Typically the user configures the `lpuart_user_config_t` instantiation as an 8-bit-char, no-parity, 1-stop-bit (8-n-1) with a baud rate of 9600 bps. The `lpuart_user_config_t` instantiation can be modified to configure the LPUART peripheral to a different baud rate or character transfer features. This is a code example to set up a user LPUART configuration instantiation:

```

lpuart_user_config_t lpuartConfig;
lpuartConfig.baudRate = 9600;
lpuartConfig.bitCountPerChar = kLpuart8BitsPerChar;
lpuartConfig.parityMode = kLpuartParityDisabled;
lpuartConfig.stopBitCount = kLpuartOneStopBit;

```

20.4.4 LPUART Transfers

The driver implements transmit and receive functions to transfer buffers of data by blocking and non-blocking modes.

The blocking transmit and receive functions include `LPUART_DRV_SendData()` and the `LPUART_DRV_ReceiveData()`.

The non-blocking (async) transmit and receive functions include `LPUART_DRV_SendData_async()` and the `LPUART_DRV_ReceiveData_async()`.

In all these cases, the functions are interrupt driven.

Chapter 21

Memory Protection Unit (MPU)

21.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Memory Protection Unit (MPU) block of Kinetis devices.

Modules

- [MPU HAL driver](#)
- [MPU Peripheral driver](#)

21.2 MPU HAL driver

21.2.1 Overview

This section describes the programming interface of the MPU HAL driver.

Enumerations

- enum `mpu_region_num` {
 kMPURegionNum00 = 0U,
 kMPURegionNum01 = 1U,
 kMPURegionNum02 = 2U,
 kMPURegionNum03 = 3U,
 kMPURegionNum04 = 4U,
 kMPURegionNum05 = 5U,
 kMPURegionNum06 = 6U,
 kMPURegionNum07 = 7U,
 kMPURegionNum08 = 8U,
 kMPURegionNum09 = 9U,
 kMPURegionNum10 = 10U,
 kMPURegionNum11 = 11U }
 MPU region number register0~register11.
- enum `mpu_error_addr_reg` {
 kMPUErrorAddrReg00 = 0U,
 kMPUErrorAddrReg01 = 1U,
 kMPUErrorAddrReg02 = 2U,
 kMPUErrorAddrReg03 = 3U,
 kMPUErrorAddrReg04 = 4U }
 MPU error address register0~4.
- enum `mpu_error_detail_reg` {
 kMPUErrorDetailReg00 = 0U,
 kMPUErrorDetailReg01 = 1U,
 kMPUErrorDetailReg02 = 2U,
 kMPUErrorDetailReg03 = 3U,
 kMPUErrorDetailReg04 = 4U }
 MPU error detail register0~4.
- enum `mpu_error_access_type` {
 kMPUReadErrorType = 0U,
 kMPUWriteErrorType = 1U }
 MPU access error.
- enum `mpu_error_attributes` {
 kMPUUserModeInstructionAccess = 0U,
 kMPUUserModeDataAccess = 1U,
 kMPUSupervisorModeInstructionAccess = 2U,
 kMPUSupervisorModeDataAccess = 3U }

- *MPU access error attributes.*
- enum `mpu_access_mode` {

 `kMPUAccessInUserMode` = 0U,

 `kMPUAccessInSupervisorMode` = 1U }

access MPU in which mode.
- enum `mpu_master_num` {

 `kMPUMaster00` = 0U,

 `kMPUMaster01` = 1U,

 `kMPUMaster02` = 2U,

 `kMPUMaster03` = 3U,

 `kMPUMaster04` = 4U,

 `kMPUMaster05` = 5U,

 `kMPUMaster06` = 6U,

 `kMPUMaster07` = 7U }

MPU master number.
- enum `mpu_error_access_control` {

 `kMPUNoRegionHitError` = 0U,

 `kMPUNoneOverlappRegionError` = 1U,

 `kMPUOverlappRegionError` = 2U }

MPU error access control detail.
- enum `mpu_supervisor_access_rights` {

 `kMPUSupervisorReadWriteExecute` = 0U,

 `kMPUSupervisorReadExecute` = 1U,

 `kMPUSupervisorReadWrite` = 2U,

 `kMPUSupervisorEqualToUsermode` = 3U }

MPU access rights in supervisor mode for master0~master3.
- enum `mpu_user_access_rights` {

 `kMPUUserNoAccessRights` = 0U,

 `kMPUUserExecute` = 1U,

 `kMPUUserWrite` = 2U,

 `kMPUUserWriteExecute` = 3U,

 `kMPUUserRead` = 4U,

 `kMPUUserReadExecute` = 5U,

 `kMPUUserReadWrite` = 6U,

 `kMPUUserReadWriteExecute` = 7U }

MPU access rights in user mode for master0~master3.
- enum `mpu_access_control` {

 `kMPUAccessDisable` = 0U,

 `kMPUAccessEnable` = 1U }

MPU access control for master4~master7.
- enum `mpu_access_type` {

 `kMPUAccessRead` = 0U,

 `kMPUAccessWrite` = 1U }

MPU access type for master4~master7.
- enum `mpu_region_valid` {

 `kMPURegionInvalid` = 0U,

 `kMPURegionValid` = 1U }

MPU HAL driver

- *MPU access region valid.*
 - enum `mpu_status_t` {
 `kStatus_MPU_Success` = 0x0U,
 `kStatus_MPU_NotInitialized` = 0x1U,
 `kStatus_MPU_NullArgument` = 0x2U }
- MPU status return codes.*

MPU HAL.

- static void `MPU_HAL_Enable` (uint32_t baseAddr)
Enables the MPU module operation.
- static void `MPU_HAL_Disable` (uint32_t baseAddr)
Disables the MPU module operation.
- static bool `MPU_HAL_IsEnabled` (uint32_t baseAddr)
Checks whether the MPU module is enabled.
- static uint32_t `MPU_HAL_GetNumberOfRegions` (uint32_t baseAddr)
Returns the total region number.
- static uint32_t `MPU_HAL_GetNumberOfSlaves` (uint32_t baseAddr)
Returns MPU slave sports.
- static uint32_t `MPU_HAL_GetHardwareRevisionLevel` (uint32_t baseAddr)
Returns hardware level info.
- static uint32_t `MPU_HAL_GetErrorAccessAddr` (uint32_t baseAddr, `mpu_error_addr_reg` regNum)
Returns hardware level info.
- static uint8_t `MPU_HAL_GetErrorSlaveSports` (uint32_t baseAddr)
Returns error access slaves sports.
- static `mpu_error_access_type` `MPU_HAL_GetErrorAccessType` (uint32_t baseAddr, `mpu_error_detail_reg` errorDetailRegNum)
Returns error access address.
- static `mpu_error_attributes` `MPU_HAL_GetErrorAttributes` (uint32_t baseAddr, `mpu_error_detail_reg` errorDetailRegNum)
Returns error access attributes.
- static `mpu_master_num` `MPU_HAL_GetErrorMasterNum` (uint32_t baseAddr, `mpu_error_detail_reg` errorDetailRegNum)
Returns error access master number.
- static `mpu_error_access_control` `MPU_HAL_GetErrorAccessControl` (uint32_t baseAddr, `mpu_error_detail_reg` errorDetailRegNum)
Returns error access control.
- static uint32_t `MPU_HAL_GetRegionStartAddr` (uint32_t baseAddr, `mpu_region_num` regionNum)
Returns the region start address.
- static void `MPU_HAL_SetRegionStartAddr` (uint32_t baseAddr, `mpu_region_num` regionNum, uint32_t startAddr)
Sets region start address.
- static uint32_t `MPU_HAL_GetRegionEndAddr` (uint32_t baseAddr, `mpu_region_num` regionNum)
Returns region end address.
- static void `MPU_HAL_SetRegionEndAddr` (uint32_t baseAddr, `mpu_region_num` regionNum, uint32_t endAddr)

- static uint32_t **MPU_HAL_GetAllMastersAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Sets region end address.

Returns all masters access permission for a specific region.
- static void **MPU_HAL_SetAllMastersAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, uint32_t accessRights)

Sets all masters access permission for a specific region.
- static mpu_supervisor_access_rights **MPU_HAL_GetM0SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M0 access permission in supervisor mode.
- static mpu_user_access_rights **MPU_HAL_GetM0UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M0 access permission in user mode.
- static void **MPU_HAL_SetM0SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights)

Sets M0 access permission in supervisor mode.
- static void **MPU_HAL_SetM0UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)

Sets M0 access permission in user mode.
- static mpu_supervisor_access_rights **MPU_HAL_GetM1SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M1 access permission in supervisor mode.
- static mpu_user_access_rights **MPU_HAL_GetM1UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M1 access permission in user mode.
- static void **MPU_HAL_SetM1SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights)

Sets M1 access permission in supervisor mode.
- static void **MPU_HAL_SetM1UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)

Sets M1 access permission in user mode.
- static mpu_supervisor_access_rights **MPU_HAL_GetM2SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M2 access permission in supervisor mode.
- static mpu_user_access_rights **MPU_HAL_GetM2UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M2 access permission in user mode.
- static void **MPU_HAL_SetM2SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights)

Sets M2 access permission in supervisor mode.
- static void **MPU_HAL_SetM2UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)

Sets M2 access permission in user mode.
- static mpu_supervisor_access_rights **MPU_HAL_GetM3SupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M3 access permission in supervisor mode.
- static mpu_user_access_rights **MPU_HAL_GetM3UserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)

Gets M3 access permission in user mode.

MPU HAL driver

- static void `MPU_HAL_SetM3SupervisorAccessRights` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_supervisor_access_rights` accessRights)
Sets M3 access permission in supervisor mode.
- static void `MPU_HAL_SetM3UserAccessRights` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_user_access_rights` accessRights)
Sets M3 access permission in user mode.
- static `mpu_access_control` `MPU_HAL_GetM4AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType)
Gets the M4 access permission.
- static void `MPU_HAL_SetM4AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType, `mpu_access_control` accessControl)
Sets the M4 access permission.
- static `mpu_access_control` `MPU_HAL_GetM5AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType)
Gets the M5 access permission.
- static void `MPU_HAL_SetM5AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType, `mpu_access_control` accessControl)
Sets the M5 access permission.
- static `mpu_access_control` `MPU_HAL_GetM6AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType)
Gets the M6 access permission.
- static void `MPU_HAL_SetM6AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType, `mpu_access_control` accessControl)
Sets the M6 access permission.
- static `mpu_access_control` `MPU_HAL_GetM7AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType)
Gets the M7 access permission.
- static void `MPU_HAL_SetM7AccessControl` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_access_type` accessType, `mpu_access_control` accessControl)
Sets the M7 access permission.
- static bool `MPU_HAL_IsRegionValid` (uint32_t baseAddr, `mpu_region_num` regionNum)
Checks whether region is valid.
- static void `MPU_HAL_SetRegionValidValue` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_region_valid` validValue)
Sets the region valid value.
- static uint32_t `MPU_HAL_GetAllMastersAlternateAcessRights` (uint32_t baseAddr, `mpu_region_num` regionNum)
Gets all masters access permission from alternative register.
- static void `MPU_HAL_SetAllMastersAlternateAccessRights` (uint32_t baseAddr, `mpu_region_num` regionNum, uint32_t accessRights)
Sets all masters access permission through alternative register.
- static `mpu_supervisor_access_rights` `MPU_HAL_GetM0AlternateSupervisorAccessRights` (uint32_t baseAddr, `mpu_region_num` regionNum)
Gets the M0 access rights in supervisor mode.
- static `mpu_user_access_rights` `MPU_HAL_GetM0AlternateUserAccessRights` (uint32_t baseAddr, `mpu_region_num` regionNum)
Gets the M0 access rights in user mode.
- static void `MPU_HAL_SetM0AlternateSupervisorAccessRights` (uint32_t baseAddr, `mpu_region_num` regionNum, `mpu_supervisor_access_rights` accessRights)

- static void **MPU_HAL_SetM0AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)
 - Sets the M0 access rights in supervisor mode.*
- static mpu_supervisor_access_rights **MPU_HAL_GetM1AlternateSupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)
 - Gets M1 access rights in supervisor mode.*
- static mpu_user_access_rights **MPU_HAL_GetM1AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)
 - Gets M1 access rights in user mode.*
- static void **MPU_HAL_SetM1AlternateSupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights)
 - Sets M1 access rights in supervisor mode.*
- static void **MPU_HAL_SetM1AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)
 - Sets M1 access rights in user mode.*
- static mpu_supervisor_access_rights **MPU_HAL_GetM2AlternateSupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)
 - Gets M2 access rights in supervisor mode.*
- static mpu_user_access_rights **MPU_HAL_GetM2AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)
 - Gets M2 access rights in user mode.*
- static void **MPU_HAL_SetM2AlternateSupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights)
 - Sets M2 access rights in supervisor mode.*
- static void **MPU_HAL_SetM2AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)
 - Sets M2 access rights in user mode.*
- static mpu_supervisor_access_rights **MPU_HAL_GetM3AlternateSupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)
 - Gets M3 access rights in supervisor mode.*
- static mpu_user_access_rights **MPU_HAL_GetM3AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum)
 - Gets M3 access rights in user mode.*
- static void **MPU_HAL_SetM3AlternateSupervisorAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights)
 - Sets M3 access rights in supervisor mode.*
- static void **MPU_HAL_SetM3AlternateUserAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights)
 - Sets M3 access rights in user mode.*
- static mpu_access_control **MPU_HAL_GetM4AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType)
 - Gets M4 access permission from alternate register.*
- static void **MPU_HAL_SetM4AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType, mpu_access_control accessControl)
 - Sets M4 access permission through alternate register.*
- static mpu_access_control **MPU_HAL_GetM5AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType)
 - Gets M5 access permission from alternate register.*

MPU HAL driver

- static void **MPU_HAL_SetM5AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType, mpu_access_control accessControl)
Sets M5 access permission through alternate register.
- static mpu_access_control **MPU_HAL_GetM6AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType)
Gets M6 access permission from alternate register.
- static void **MPU_HAL_SetM6AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType, mpu_access_control accessControl)
Sets M6 access permission through alternate register.
- static mpu_access_control **MPU_HAL_GetM7AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType)
Gets M7 access permission from alternate register.
- static void **MPU_HAL_SetM7AlternateAccessRights** (uint32_t baseAddr, mpu_region_num regionNum, mpu_access_type accessType, mpu_access_control accessControl)
Sets M7 access permission through alternate register.
- void **MPU_HAL_Init** (uint32_t baseAddr)
Initializes the MPU module.

21.2.2 Enumeration Type Documentation

21.2.2.1 enum mpu_region_num

Enumerator

- kMPURegionNum00** MPU region number 0.
kMPURegionNum01 MPU region number 1.
kMPURegionNum02 MPU region number 2.
kMPURegionNum03 MPU region number 3.
kMPURegionNum04 MPU region number 4.
kMPURegionNum05 MPU region number 5.
kMPURegionNum06 MPU region number 6.
kMPURegionNum07 MPU region number 7.
kMPURegionNum08 MPU region number 8.
kMPURegionNum09 MPU region number 9.
kMPURegionNum10 MPU region number 10.
kMPURegionNum11 MPU region number 11.

21.2.2.2 enum mpu_error_addr_reg

Enumerator

- kMPUErrorAddrReg00** MPU error address register 0.
kMPUErrorAddrReg01 MPU error address register 1.
kMPUErrorAddrReg02 MPU error address register 2.
kMPUErrorAddrReg03 MPU error address register 3.

kMPUErrorAddrReg04 MPU error address register 4.

21.2.2.3 enum mpu_error_detail_reg

Enumerator

kMPUErrorDetailReg00 MPU error detail register 0.
kMPUErrorDetailReg01 MPU error detail register 1.
kMPUErrorDetailReg02 MPU error detail register 2.
kMPUErrorDetailReg03 MPU error detail register 3.
kMPUErrorDetailReg04 MPU error detail register 4.

21.2.2.4 enum mpu_error_access_type

Enumerator

kMPUReadErrorType MPU error type—read.
kMPUWriteErrorType MPU error type—write.

21.2.2.5 enum mpu_error_attributes

Enumerator

kMPUUserModeInstructionAccess access instruction error in user mode
kMPUUserModeDataAccess access data error in user mode
kMPUSupervisorModeInstructionAccess access instruction error in supervisor mode
kMPUSupervisorModeDataAccess access data error in supervisor mode

21.2.2.6 enum mpu_access_mode

Enumerator

kMPUAccessInUserMode access data or instruction in user mode
kMPUAccessInSupervisorMode access data or instruction in supervisor mode

21.2.2.7 enum mpu_master_num

Enumerator

kMPUMaster00 Core.
kMPUMaster01 Debugger.

MPU HAL driver

kMPUMaster02 DMA.
kMPUMaster03 ENET.
kMPUMaster04 USB.
kMPUMaster05 SDHC.
kMPUMaster06 undefined.
kMPUMaster07 undefined.

21.2.2.8 enum mpu_error_access_control

Enumerator

kMPUNoRegionHitError no region hit error
kMPUNoneOverlappRegionError access single region error
kMPUOverlappRegionError access overlapping region error

21.2.2.9 enum mpu_supervisor_access_rights

Enumerator

kMPUSupervisorReadWriteExecute R W E allowed in supervisor mode.
kMPUSupervisorReadExecute R E allowed in supervisor mode.
kMPUSupervisorReadWrite R W allowed in supervisor mode.
kMPUSupervisorEqualToUsermode access permission equal to user mode

21.2.2.10 enum mpu_user_access_rights

Enumerator

kMPUUserNoAccessRights no access allowed in user mode
kMPUUserExecute E allowed in user mode.
kMPUUserWrite W allowed in user mode.
kMPUUserWriteExecute W E allowed in user mode.
kMPUUserRead R allowed in user mode.
kMPUUserReadExecute R E allowed in user mode.
kMPUUserReadWrite R W allowed in user mode.
kMPUUserReadWriteExecute R W E allowed in user mode.

21.2.2.11 enum mpu_access_control

Enumerator

kMPUAccessDisable Read or Write not allowed.
kMPUAccessEnable Read or Write allowed.

21.2.2.12 enum mpu_access_type

Enumerator

kMPUAccessRead Access type is read.

kMPUAccessWrite Access type is write.

21.2.2.13 enum mpu_region_valid

Enumerator

kMPURegionInvalid region invalid

kMPURegionValid region valid

21.2.2.14 enum mpu_status_t

Enumerator

kStatus_MPU_Success Succeed.

kStatus_MPU_NotInitialized MPU is not initialized yet.

kStatus_MPU_NullArgument Argument is NULL.

21.2.3 Function Documentation

21.2.3.1 static void MPU_HAL_Enable(uint32_t baseAddr) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

21.2.3.2 static void MPU_HAL_Disable(uint32_t baseAddr) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

21.2.3.3 static bool MPU_HAL_IsEnabled(uint32_t baseAddr) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

Returns

true MPU module is enabled
false MPU module is disabled

21.2.3.4 static uint32_t MPU_HAL_GetNumberOfRegions (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

Returns

the number of regions

21.2.3.5 static uint32_t MPU_HAL_GetNumberOfSlaves (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

Returns

the number of slaves

21.2.3.6 static uint32_t MPU_HAL_GetHardwareRevisionLevel (*uint32_t baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

Returns

hardware revision level

21.2.3.7 static uint32_t MPU_HAL_GetErrorAccessAddr (uint32_t *baseAddr*, mpu_error_addr_reg *regNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regNum</i>	Error address register number

Returns

error access address

21.2.3.8 static uint8_t MPU_HAL_GetErrorSlaveSports (uint32_t *baseAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

Returns

error slave sports

21.2.3.9 static mpu_error_access_type MPU_HAL_GetErrorAccessType (uint32_t *baseAddr*, mpu_error_detail_reg *errorDetailRegNum*) [inline], [static]

Parameters

MPU HAL driver

<i>baseAddr</i>	The MPU peripheral base address
<i>errorDetail-RegNum</i>	Error detail register number

Returns

error access type

21.2.3.10 static mpu_error_attributes MPU_HAL_GetErrorAttributes (uint32_t *baseAddr*, mpu_error_detail_reg *errorDetailRegNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>errorDetail-RegNum</i>	Detail error register number

Returns

error access attributes

21.2.3.11 static mpu_master_num MPU_HAL_GetErrorMasterNum (uint32_t *baseAddr*, mpu_error_detail_reg *errorDetailRegNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>errorDetail-RegNum</i>	Error register number

Returns

error master number

21.2.3.12 static mpu_error_access_control MPU_HAL_GetErrorAccessControl (uint32_t *baseAddr*, mpu_error_detail_reg *errorDetailRegNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>errorDetail-RegNum</i>	Error register number

Returns

error access control

21.2.3.13 static uint32_t MPU_HAL_GetRegionStartAddr (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

region start address

21.2.3.14 static void MPU_HAL_SetRegionStartAddr (uint32_t *baseAddr*, mpu_region_num *regionNum*, uint32_t *startAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>startAddr</i>	Region start address

21.2.3.15 static uint32_t MPU_HAL_GetRegionEndAddr (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

region end address

21.2.3.16 static void MPU_HAL_SetRegionEndAddr (uint32_t *baseAddr*, mpu_region_num *regionNum*, uint32_t *endAddr*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>endAddr</i>	Region end address

21.2.3.17 static uint32_t MPU_HAL_GetAllMastersAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

all masters access permission

21.2.3.18 static void MPU_HAL_SetAllMastersAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, uint32_t *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	All masters access rights

21.2.3.19 static mpu_supervisor_access_rights MPU_HAL_GetM0SupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master0 access permission

21.2.3.20 static mpu_user_access_rights MPU_HAL_GetM0UserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master0 access permission

21.2.3.21 static void MPU_HAL_SetM0SupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_supervisor_access_rights *accessRights*) [inline], [static]

Parameters

MPU HAL driver

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master0 access permission

21.2.3.22 static void MPU_HAL_SetM0UserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_user_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master0 access permission

21.2.3.23 static mpu_supervisor_access_rights MPU_HAL_GetM1SupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master1 access permission

21.2.3.24 static mpu_user_access_rights MPU_HAL_GetM1UserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
-----------------	---------------------------------

<i>regionNum</i>	MPU region number
------------------	-------------------

Returns

Master1 access permission

21.2.3.25 static void MPU_HAL_SetM1SupervisorAccessRights (*uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master1 access permission

21.2.3.26 static void MPU_HAL_SetM1UserAccessRights (*uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master1 access permission

21.2.3.27 static mpu_supervisor_access_rights MPU_HAL_GetM2SupervisorAccessRights (*uint32_t baseAddr, mpu_region_num regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master2 access permission

**21.2.3.28 static mpu_user_access_rights MPU_HAL_GetM2UserAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*) [inline], [static]**

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master2 access permission

21.2.3.29 static void MPU_HAL_SetM2SupervisorAccessRights (*uint32_t baseAddr, mpu_region_num regionNum, mpu_supervisor_access_rights accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master2 access permission

21.2.3.30 static void MPU_HAL_SetM2UserAccessRights (*uint32_t baseAddr, mpu_region_num regionNum, mpu_user_access_rights accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master2 access permission

21.2.3.31 static mpu_supervisor_access_rights MPU_HAL_GetM3SupervisorAccessRights (*uint32_t baseAddr, mpu_region_num regionNum*) [inline], [static]

Parameters

MPU HAL driver

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master3 access permission

21.2.3.32 static mpu_user_access_rights MPU_HAL_GetM3UserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master3 access permission

21.2.3.33 static void MPU_HAL_SetM3SupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_supervisor_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessRights</i>	Master3 access permission.

21.2.3.34 static void MPU_HAL_SetM3UserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_user_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master3 access permission

21.2.3.35 static mpu_access_control MPU_HAL_GetM4AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write

Returns

read or write permission

21.2.3.36 static void MPU_HAL_SetM4AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*, mpu_access_control *accessControl*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write
<i>accessControl</i>	Access permission

21.2.3.37 static mpu_access_control MPU_HAL_GetM5AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write

Returns

read or write permission

21.2.3.38 static void MPU_HAL_SetM5AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*, mpu_access_control *accessControl*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write
<i>accessControl</i>	Access permission

21.2.3.39 static mpu_access_control MPU_HAL_GetM6AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	access type Read/Write

Returns

read or write permission

21.2.3.40 static void MPU_HAL_SetM6AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*, mpu_access_control *accessControl*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write
<i>accessControl</i>	Access permission

21.2.3.41 static mpu_access_control MPU_HAL_GetM7AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write

Returns

read or write permission

21.2.3.42 static void MPU_HAL_SetM7AccessControl (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*, mpu_access_control *accessControl*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessType</i>	Access type Read/Write
<i>accessControl</i>	Access permission

21.2.3.43 static bool MPU_HAL_IsRegionValid (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

true region is valid
false region is invalid

21.2.3.44 static void MPU_HAL_SetRegionValidValue (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_region_valid *validValue*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>validValue</i>	Region valid value

21.2.3.45 static uint32_t MPU_HAL_GetAllMastersAlternateAcessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

all masters access permission

21.2.3.46 static void MPU_HAL_SetAllMastersAlternateAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, uint32_t *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	All masters access permission

21.2.3.47 static mpu_supervisor_access_rights MPU_HAL_GetM0AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master0 access permission

21.2.3.48 static mpu_user_access_rights MPU_HAL_GetM0AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master0 access permission

21.2.3.49 static void MPU_HAL_SetM0AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_supervisor_access_rights *accessRights*) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master0 access permission

21.2.3.50 static void MPU_HAL_SetM0AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_user_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master0 access permission

21.2.3.51 static mpu_supervisor_access_rights MPU_HAL_GetM1AlternateSupervisor-AccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master1 access permission

21.2.3.52 static mpu_user_access_rights MPU_HAL_GetM1AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

Master1 access permission

21.2.3.53 static void MPU_HAL_SetM1AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_supervisor_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master1 access permission

21.2.3.54 static void MPU_HAL_SetM1AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_user_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master1 access permission

21.2.3.55 static mpu_supervisor_access_rights MPU_HAL_GetM2AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

M2 access permission

21.2.3.56 static mpu_user_access_rights MPU_HAL_GetM2AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

M2 access permission

21.2.3.57 static void MPU_HAL_SetM2AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_supervisor_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	M2 access permission

21.2.3.58 static void MPU_HAL_SetM2AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_user_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	M2 access permission

21.2.3.59 static mpu_supervisor_access_rights MPU_HAL_GetM3AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

M3 access permission

21.2.3.60 static mpu_user_access_rights MPU_HAL_GetM3AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number

Returns

M3 access permission

21.2.3.61 static void MPU_HAL_SetM3AlternateSupervisorAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_supervisor_access_rights *accessRights*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master3 access permission

21.2.3.62 static void MPU_HAL_SetM3AlternateUserAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_user_access_rights *accessRights*) [inline], [static]

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address
<i>regionNum</i>	MPU region number
<i>accessRights</i>	Master3 access permission

**21.2.3.63 static mpu_access_control MPU_HAL_GetM4AlternateAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*, mpu_access_type *accessType*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.

Returns

read or write permission.

**21.2.3.64 static void MPU_HAL_SetM4AlternateAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*, mpu_access_type *accessType*,
mpu_access_control *accessControl*) [inline], [static]**

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.
<i>accessControl</i>	Access permission.

**21.2.3.65 static mpu_access_control MPU_HAL_GetM5AlternateAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*, mpu_access_type *accessType*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.

Returns

read or write permission.

**21.2.3.66 static void MPU_HAL_SetM5AlternateAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*, mpu_access_type *accessType*,
mpu_access_control *accessControl*) [inline], [static]**

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.
<i>accessControl</i>	Master5 Access permission.

**21.2.3.67 static mpu_access_control MPU_HAL_GetM6AlternateAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*, mpu_access_type *accessType*)
[inline], [static]**

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.

Returns

read or write permission.

**21.2.3.68 static void MPU_HAL_SetM6AlternateAccessRights (uint32_t
baseAddr, mpu_region_num *regionNum*, mpu_access_type *accessType*,
mpu_access_control *accessControl*) [inline], [static]**

MPU HAL driver

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.
<i>accessControl</i>	Master6 access permission.

21.2.3.69 static mpu_access_control MPU_HAL_GetM7AlternateAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.

Returns

read or write permission.

21.2.3.70 static void MPU_HAL_SetM7AlternateAccessRights (uint32_t *baseAddr*, mpu_region_num *regionNum*, mpu_access_type *accessType*, mpu_access_control *accessControl*) [inline], [static]

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
<i>regionNum</i>	MPU region number.
<i>accessType</i>	Access type Read/Write.
<i>accessControl</i>	Master7 access permission.

21.2.3.71 void MPU_HAL_Init (uint32_t *baseAddr*)

Parameters

<i>baseAddr</i>	The MPU peripheral base address.
-----------------	----------------------------------

MPU Peripheral driver

21.3 MPU Peripheral driver

21.3.1 Overview

This section describes the programming interface of the MPU Peripheral driver. The MPU driver configures MPU.

21.3.2 MPU Initialization

To initialize the MPU module, call the [MPU_DRV_Init\(\)](#) function and provide the user configuration data structure. This function sets the configuration of the MPU module automatically and enables the MPU module.

Note that the configuration start address, end address, the region valid value and the debugger's access permission for the MPU region 0 cannot be changed.

This is example code to configure the MPU driver:

```
// Define MPU memory access permission configuration structure . //
struct mpu_access_rights_t mpuAccessRights{
    .m0UserMode          = kMPUUserNoAccessRights;
    .m0SupervisorMode   = kMPUSupervisorReadWriteExecute;
    .m0Process_identifier = kMPUIIdentifierDisable;
    .m1UserMode          = kMPUUserNoAccessRights;
    .m1SupervisorMode   = kMPUSupervisorEqualToUsermode;
    .m1Process_identifier = kMPUIIdentifierDisable;
    .m2UserMode          = kMPUUserNoAccessRights;
    .m2SupervisorMode   = kMPUSupervisorEqualToUsermode;
    .m2Process_identifier = kMPUIIdentifierDisable;
    .m3UserMode          = kMPUUserNoAccessRights;
    .m3SupervisorMode   = kMPUSupervisorEqualToUsermode;
    .m3Process_identifier = kMPUIIdentifierDisable;
    .m4WriteControl      = kMPUAccessDisable;
    .m4ReadControl       = kMPUAccessDisable;
    .m5WriteControl      = kMPUAccessDisable;
    .m5ReadControl       = kMPUAccessDisable;
    .m6WriteControl      = kMPUAccessDisable;
    .m6ReadControl       = kMPUAccessDisable;
    .m7WriteControl      = kMPUAccessDisable;
    .m7ReadControl       = kMPUAccessDisable;
};

// Defines MPU region configuration structure . //
struct mpu_region_config_t mpuRegionConfig{
    .regionNum        = kMPURegionNum00;
    .startAddr        = 0x0;
    .endAddr          = 0xffffffff;
    .accessRights     = mpuAccessRights;
};

// Defines MPU user configuration structure . //
struct mpu_user_config_t mpuUserConfig{
    .regionConfig     = mpuRegionConfig;
    .next             = NULL;
}mpu_user_config_t;

// Initializes MPU region 0. //
MPU_DRV_Init(0, &mpuUserConfig);
```

21.3.3 MPU Interrupt

1. The interrupt corresponding to BUSFAULT causes an error by accessing the core.
2. Definition for the MPU IRQ function.

```
void MPU_DRV_IRQHandler(uint32_t instance)
{
    assert(instance < HW_MPU_INSTANCE_COUNT);

    if (mpu_state_ptrs[instance])
    {
        if (mpu_state_ptrs[instance]->userCallbackFunc)
        {
            // Execute user-defined callback function. //
            (* (mpu_state_ptrs[instance]->userCallbackFunc))();
        }
    }
}
```

Data Structures

- struct [mpu_access_rights_t](#)
Data c for the MPU region access permission initialize. [More...](#)
- struct [mpu_region_config_t](#)
Data v for MPU region initialize. [More...](#)
- struct [mpu_user_config_t](#)
Data The chapter describes the programming interface of the for MPU region initialization. [More...](#)

Variables

- const uint32_t [g_mpuBaseAddr](#) []
Table of base addresses for MPU instances.
- const IRQn_Type [g_mpuIrqId](#) [HW_MPU_INSTANCE_COUNT]
Table to save MPU IRQ enumeration numbers defined in the CMSIS header file.

MPU Driver

MPU driver user call back function.

The contents of this structure provides a callback function.

- [mpu_status_t MPU_DRV_Init](#) (uint32_t instance, [mpu_user_config_t](#) *userConfigPtr)
Initializes the MPU driver.
- [mpu_status_t MPU_DRV_RegionInit](#) (uint32_t instance, [mpu_region_config_t](#) *regionConfigPtr)
Initializes the MPU region.
- [mpu_status_t MPU_DRV_Deinit](#) (uint32_t instance)
Deinitializes the MPU region.
- [mpu_status_t MPU_DRV_SetRegionAccessPermission](#) (uint32_t instance, [mpu_region_num](#) regionNum, [mpu_access_rights_t](#) accessRights)
Configures the MPU region access permission.

MPU Peripheral driver

- `mpu_access_rights_t MPU_DRV_GetRegionAccessPermission` (`uint32_t` instance, `mpu_region_num` `regionNum`)
Gets the MPU region access permission.
- `bool MPU_DRV_IsRegionValid` (`uint32_t` instance, `mpu_region_num` `regionNum`)
Checks whether the MPU region is valid.
- `mpu_status_t MPU_DRV_SetRegionValid` (`uint32_t` instance, `mpu_region_num` `regionNum`)
Sets the MPU region valid.

21.3.4 Data Structure Documentation

21.3.4.1 struct mpu_access_rights_t

This structure is used when calling the `MPU_DRV_Init` function.

Data Fields

- `uint32_t m0UserMode: 3`
master0 access permission in user mode
- `uint32_t m0SupervisorMode: 2`
master0 access permission in supervisor mode
- `uint32_t m1UserMode: 3`
master1 access permission in user mode
- `uint32_t m1SupervisorMode: 2`
master1 access permission in supervisor mode
- `uint32_t m2UserMode: 3`
master2 access permission in user mode
- `uint32_t m2SupervisorMode: 2`
master2 access permission in supervisor mode
- `uint32_t m3UserMode: 3`
master3 access permission in user mode
- `uint32_t m3SupervisorMode: 2`
master3 access permission in supervisor mode
- `uint32_t m4WriteControl: 1`
master4 write access permission
- `uint32_t m4ReadControl: 1`
master4 read access permission
- `uint32_t m5WriteControl: 1`
master5 write access permission
- `uint32_t m5ReadControl: 1`
master5 read access permission
- `uint32_t m6WriteControl: 1`
master6 write access permission
- `uint32_t m6ReadControl: 1`
master6 read access permission
- `uint32_t m7WriteControl: 1`
master7 write access permission
- `uint32_t m7ReadControl: 1`
master7 read access permission

21.3.4.2 struct mpu_region_config_t

This structure is used when calling the MPU_DRV_Init function.

Data Fields

- `mpu_region_num regionNum`
MPU region number.
- `uint32_t startAddr`
memory region start address
- `uint32_t endAddr`
memory region end address
- `mpu_access_rights_t accessRights`
all masters access permission

21.3.4.3 struct mpu_user_config_t

This structure is used when calling the MPU_DRV_Init function.

Data Fields

- `mpu_region_config_t regionConfig`
region access permission
- `struct MpuUserConfig * next`
pointer to the next structure

21.3.5 Function Documentation

21.3.5.1 `mpu_status_t MPU_DRV_Init (uint32_t instance, mpu_user_config_t * userConfigPtr)`

Parameters

<code>instance</code>	The MPU peripheral instance number.
<code>userConfigPtr</code>	The pointer to the MPU user configure structure, see mpu_user_config_t .
<code>userStatePtr</code>	The pointer of run time structure.

Returns

`kStatus_MPU_Success` means success. Otherwise, means failure.

**21.3.5.2 mpu_status_t MPU_DRV_RegionInit (uint32_t *instance*, mpu_region_config_t *
regionConfigPtr)**

Parameters

<i>instance</i>	The MPU peripheral instance number.
<i>regionConfig-Ptr</i>	The pointer to the MPU user configure structure, see mpu_user_config_t .

Returns

kStatus_MPU_Success means success. Otherwise, means failure.

21.3.5.3 mpu_status_t MPU_DRV_Deinit (uint32_t *instance*)

Parameters

<i>instance</i>	The MPU peripheral instance number.
-----------------	-------------------------------------

Returns

kStatus_MPU_Success means success. Otherwise, means failure.

21.3.5.4 mpu_status_t MPU_DRV_SetRegionAccessPermission (uint32_t *instance*, mpu_region_num *regionNum*, mpu_access_rights_t *accessRights*)

Parameters

<i>instance</i>	The MPU peripheral instance number.
<i>regionNum</i>	The MPU region number.
<i>accessRights</i>	A pointer to access permission structure.

Returns

kStatus_MPU_Success means success. Otherwise, means failure.

21.3.5.5 mpu_access_rights_t MPU_DRV_GetRegionAccessPermission (uint32_t *instance*, mpu_region_num *regionNum*)

MPU Peripheral driver

Parameters

<i>instance</i>	The MPU peripheral instance number.
<i>regionNum</i>	The MPU region number.

Returns

access permission.

21.3.5.6 **bool MPU_DRV_IsRegionValid (uint32_t *instance*, mpu_region_num *regionNum*)**

Parameters

<i>instance</i>	The MPU peripheral instance number.
<i>regionNum</i>	MPU region number.

Returns

bool value.

21.3.5.7 **mpu_status_t MPU_DRV_SetRegionValid (uint32_t *instance*, mpu_region_num *regionNum*)**

Parameters

<i>instance</i>	The MPU peripheral instance number.
<i>regionNum</i>	MPU region number.

21.3.6 Variable Documentation

21.3.6.1 **const uint32_t g_mpuBaseAddr[]**

21.3.6.2 **const IRQn_Type g_mpulrqld[HW_MPU_INSTANCE_COUNT]**

Chapter 22

Programmable Delay Block (PDB)

22.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Programmable Delay Block (PDB) block of Kinetis devices.

Modules

- [PDB HAL driver](#)
- [PDB Peripheral driver](#)

This part describes the programming interface of the PDB Peripheral driver.

22.2 PDB HAL driver

22.2.1 Overview

This section describes the programming interface of the PDB HAL driver.

Enumerations

- enum `pdb_status_t` {
 `kStatus_PDB_Success` = 0U,
 `kStatus_PDB_InvalidArgument` = 1U,
 `kStatus_PDB_Failed` = 2U }
 PDB status return codes.
- enum `pdb_load_mode_t` {
 `kPdbLoadImmediately` = 0U,
 `kPdbLoadAtModuloCounter` = 1U,
 `kPdbLoadAtNextTrigger` = 2U,
 `kPdbLoadAtModuloCounterOrNextTrigger` = 3U }
 Defines the type of value load mode for the PDB module.
- enum `pdb_clk_prescaler_div_mode_t` {
 `kPdbClkPreDivBy1` = 0U,
 `kPdbClkPreDivBy2` = 1U,
 `kPdbClkPreDivBy4` = 2U,
 `kPdbClkPreDivBy8` = 3U,
 `kPdbClkPreDivBy16` = 4U,
 `kPdbClkPreDivBy32` = 5U,
 `kPdbClkPreDivBy64` = 6U,
 `kPdbClkPreDivBy128` = 7U }
 Defines the type of prescaler divider for the PDB counter clock.
- enum `pdb_trigger_src_mode_t` {
 `kPdbTrigger0` = 0U,
 `kPdbTrigger1` = 1U,
 `kPdbTrigger2` = 2U,
 `kPdbTrigger3` = 3U,
 `kPdbTrigger4` = 4U,
 `kPdbTrigger5` = 5U,
 `kPdbTrigger6` = 6U,
 `kPdbTrigger7` = 7U,
 `kPdbTrigger8` = 8U,
 `kPdbTrigger9` = 9U,
 `kPdbTrigger10` = 10U,
 `kPdbTrigger11` = 11U,
 `kPdbTrigger12` = 12U,
 `kPdbTrigger13` = 13U,
 `kPdbTrigger14` = 14U,

- `kPdbSoftTrigger = 15U }`
- Defines the type of trigger source mode for the PDB.*
- enum `pdb_mult_factor_mode_t` {
- `kPdbMultFactorAs1 = 0U,`
 - `kPdbMultFactorAs10 = 1U,`
 - `kPdbMultFactorAs20 = 2U,`
 - `kPdbMultFactorAs40 = 3U }`
- Defines the type of the multiplication source mode for PDB.*

Functions

- void `PDB_HAL_Init` (uint32_t baseAddr)
Resets the PDB registers to a known state.
- static void `PDB_HAL_SetLoadMode` (uint32_t baseAddr, `pdb_load_mode_t` mode)
Sets the load mode for timing registers.
- static void `PDB_HAL_SetSeqErrIntCmd` (uint32_t baseAddr, bool enable)
Switches to enable the PDB sequence error interrupt.
- static void `PDB_HAL_SetSoftTriggerCmd` (uint32_t baseAddr)
Triggers the DAC by software if enabled.
- static void `PDB_HAL_SetDmaCmd` (uint32_t baseAddr, bool enable)
Switches to enable the PDB DMA support.
- static void `PDB_HAL_SetPreDivMode` (uint32_t baseAddr, `pdb_clk_prescaler_div_mode_t` mode)
Sets the prescaler divider from the peripheral bus clock for the PDB.
- static void `PDB_HAL_SetTriggerSrcMode` (uint32_t baseAddr, `pdb_trigger_src_mode_t` mode)
Sets the trigger source mode for the PDB module.
- static void `PDB_HAL_Enable` (uint32_t baseAddr)
Switches on to enable the PDB module.
- static void `PDB_HAL_Disable` (uint32_t baseAddr)
Switches off to enable the PDB module.
- static bool `PDB_HAL_GetIntFlag` (uint32_t baseAddr)
Gets the PDB delay interrupt flag.
- static void `PDB_HAL_ClearIntFlag` (uint32_t baseAddr)
Clears the PDB delay interrupt flag.
- static void `PDB_HAL_SetIntCmd` (uint32_t baseAddr, bool enable)
Switches to enable the PDB interrupt.
- static void `PDB_HAL_SetPreMultFactorMode` (uint32_t baseAddr, `pdb_mult_factor_mode_t` mode)
Sets the PDB prescaler multiplication factor.
- static void `PDB_HAL_SetContinuousModeCmd` (uint32_t baseAddr, bool enable)
Switches to enable the PDB continuous mode.
- static void `PDB_HAL_SetLoadRegsCmd` (uint32_t baseAddr)
Loads the delay registers value for the PDB module.
- static void `PDB_HAL_SetModulusValue` (uint32_t baseAddr, uint32_t value)
Sets the modulus value for the PDB module.
- static uint32_t `PDB_HAL_GetModulusValue` (uint32_t baseAddr)
Gets the modulus value for the PDB module.
- static uint32_t `PDB_HAL_GetCounterValue` (uint32_t baseAddr)
Gets the PDB counter value.
- static void `PDB_HAL_SetIntDelayValue` (uint32_t baseAddr, uint32_t value)

PDB HAL driver

Sets the interrupt delay milestone of the PDB counter.

- static uint32_t **PDB_HAL_GetIntDelayValue** (uint32_t baseAddr)

Gets the current interrupt delay milestone of the PDB counter.

- void **PDB_HAL_SetPreTriggerBackToBackCmd** (uint32_t baseAddr, uint32_t chn, uint32_t preChn, bool enable)

Switches to enable the pre-trigger back-to-back mode.

- void **PDB_HAL_SetPreTriggerOutputCmd** (uint32_t baseAddr, uint32_t chn, uint32_t preChn, bool enable)

Switches to enable the pre-trigger output.

- void **PDB_HAL_SetPreTriggerCmd** (uint32_t baseAddr, uint32_t chn, uint32_t preChn, bool enable)

Switches to enable the pre-trigger.

- static bool **PDB_HAL_GetPreTriggerFlag** (uint32_t baseAddr, uint32_t chn, uint32_t preChn)

Gets the flag which indicates whether the PDB counter has reached the pre-trigger delay value.

- void **PDB_HAL_ClearPreTriggerFlag** (uint32_t baseAddr, uint32_t chn, uint32_t preChn)

Clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.

- static bool **PDB_HAL_GetPreTriggerSeqErrFlag** (uint32_t baseAddr, uint32_t chn, uint32_t preChn)

Gets the flag which indicates whether a sequence error is detected.

- void **PDB_HAL_ClearPreTriggerSeqErrFlag** (uint32_t baseAddr, uint32_t chn, uint32_t preChn)

Clears the flag which indicates that a sequence error has been detected.

- void **PDB_HAL_SetPreTriggerDelayCount** (uint32_t baseAddr, uint32_t chn, uint32_t preChn, uint32_t value)

Sets the pre-trigger delay value.

- static void **PDB_HAL_SetDacExtTriggerInputCmd** (uint32_t baseAddr, uint32_t dacChn, bool enable)

Switches to enable the DAC external trigger input.

- static void **PDB_HAL_SetDacIntervalTriggerCmd** (uint32_t baseAddr, uint32_t dacChn, bool enable)

Switches to enable the DAC external trigger input.

- static void **PDB_HAL_SetDacIntervalValue** (uint32_t baseAddr, uint32_t dacChn, uint32_t value)

Sets the interval value for the DAC trigger.

- static uint32_t **PDB_HAL_GetDacIntervalValue** (uint32_t baseAddr, uint32_t dacChn)

Gets the interval value for the DAC trigger.

- void **PDB_HAL_SetPulseOutCmd** (uint32_t baseAddr, uint32_t pulseChn, bool enable)

Switches to enable the pulse-out trigger.

- static void **PDB_HAL_SetPulseOutDelayForHigh** (uint32_t baseAddr, uint32_t pulseChn, uint32_t value)

Sets the counter delay value for the pulse-out goes high.

- static void **PDB_HAL_SetPulseOutDelayForLow** (uint32_t baseAddr, uint32_t pulseChn, uint32_t value)

Sets the counter delay value for the pulse-out goes low.

- static void **PDB_HAL_SetChnC1Reg** (uint32_t baseAddr, uint32_t chn, uint32_t regVal)

Sets the value roughly into ChnC1 register for special time-critical case.

- static uint32_t **PDB_HAL_GetChnC1Reg** (uint32_t baseAddr, uint32_t chn)

Gets the value roughly from ChnC1 register for special time-critical case.

22.2.2 Enumeration Type Documentation

22.2.2.1 enum pdb_status_t

Enumerator

kStatus_PDB_Success Success.

kStatus_PDB_InvalidArgument Invalid argument existed.

kStatus_PDB_Failed Execution failed.

22.2.2.2 enum pdb_load_mode_t

Some timing related registers, such as the MOD, IDLY, CHnDLYm, INTx and POyDLY, buffer the setting values. Only the load operation is triggered. The setting value is loaded from a buffer and takes effect. There are four loading modes to fit different applications.

Enumerator

kPdbLoadImmediately Loaded immediately after load operation.

kPdbLoadAtModuloCounter Loaded when counter hits the modulo after load operation.

kPdbLoadAtNextTrigger Loaded when detecting an input trigger after load operation.

kPdbLoadAtModuloCounterOrNextTrigger Loaded when counter hits the modulo or detecting an input trigger after load operation.

22.2.2.3 enum pdb_clk_prescaler_div_mode_t

Enumerator

kPdbClkPreDivBy1 Counting divided by multiplication factor selected by MULT.

kPdbClkPreDivBy2 Counting divided by multiplication factor selected by 2 times ofMULT.

kPdbClkPreDivBy4 Counting divided by multiplication factor selected by 4 times ofMULT.

kPdbClkPreDivBy8 Counting divided by multiplication factor selected by 8 times ofMULT.

kPdbClkPreDivBy16 Counting divided by multiplication factor selected by 16 times ofMULT.

kPdbClkPreDivBy32 Counting divided by multiplication factor selected by 32 times ofMULT.

kPdbClkPreDivBy64 Counting divided by multiplication factor selected by 64 times ofMULT.

kPdbClkPreDivBy128 Counting divided by multiplication factor selected by 128 times ofMULT.

22.2.2.4 enum pdb_trigger_src_mode_t

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger.

PDB HAL driver

Enumerator

- kPdbTrigger0*** Select trigger-In 0.
- kPdbTrigger1*** Select trigger-In 1.
- kPdbTrigger2*** Select trigger-In 2.
- kPdbTrigger3*** Select trigger-In 3.
- kPdbTrigger4*** Select trigger-In 4.
- kPdbTrigger5*** Select trigger-In 5.
- kPdbTrigger6*** Select trigger-In 6.
- kPdbTrigger7*** Select trigger-In 7.
- kPdbTrigger8*** Select trigger-In 8.
- kPdbTrigger9*** Select trigger-In 8.
- kPdbTrigger10*** Select trigger-In 10.
- kPdbTrigger11*** Select trigger-In 11.
- kPdbTrigger12*** Select trigger-In 12.
- kPdbTrigger13*** Select trigger-In 13.
- kPdbTrigger14*** Select trigger-In 14.
- kPdbSoftTrigger*** Select software trigger.

22.2.2.5 enum pdb_mult_factor_mode_t

Selects the multiplication factor of the prescaler divider for the PDB counter clock.

Enumerator

- kPdbMultFactorAs1*** Multiplication factor is 1.
- kPdbMultFactorAs10*** Multiplication factor is 10.
- kPdbMultFactorAs20*** Multiplication factor is 20.
- kPdbMultFactorAs40*** Multiplication factor is 40.

22.2.3 Function Documentation

22.2.3.1 void PDB_HAL_Init (uint32_t baseAddr)

This function resets the PDB registers to a known state. This state is defined in a reference manual and is power on reset value.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

22.2.3.2 static void PDB_HAL_SetLoadMode (uint32_t *baseAddr*, pdb_load_mode_t *mode*) [inline], [static]

This function sets the load mode for some timing registers including MOD, IDLY, CHnDLYm, INTx and POyDLY.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode, see to "pdb_load_mode_t".

22.2.3.3 static void PDB_HAL_SetSeqErrIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the PDB sequence error interrupt.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	The switcher to assert the feature.

22.2.3.4 static void PDB_HAL_SetSoftTriggerCmd (uint32_t *baseAddr*) [inline], [static]

If enabled, this function triggers the DAC by using software.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

22.2.3.5 static void PDB_HAL_SetDmaCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the PDB DMA support.

PDB HAL driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	The switcher to assert the feature.

22.2.3.6 static void PDB_HAL_SetPreDivMode (uint32_t *baseAddr*, pdb_clk_prescaler_div_mode_t *mode*) [inline], [static]

This function sets the prescaler divider from the peripheral bus clock for the PDB.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode, see to "pdb_clk_prescaler_div_mode_t".

22.2.3.7 static void PDB_HAL_SetTriggerSrcMode (uint32_t *baseAddr*, pdb_trigger_src_mode_t *mode*) [inline], [static]

This function sets the trigger source mode for the PDB module.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode, see to "pdb_trigger_src_mode_t".

22.2.3.8 static void PDB_HAL_Enable (uint32_t *baseAddr*) [inline], [static]

This function switches on to enable the PDB module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

22.2.3.9 static void PDB_HAL_Disable (uint32_t *baseAddr*) [inline], [static]

This function switches off to enable the PDB module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

22.2.3.10 static bool PDB_HAL_GetIntFlag (uint32_t *baseAddr*) [inline], [static]

This function gets the PDB delay interrupt flag.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

Flat status, true if the flag is set.

22.2.3.11 static void PDB_HAL_ClearIntFlag (uint32_t *baseAddr*) [inline], [static]

This function clears PDB delay interrupt flag.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

Flat status, true if the flag is set.

22.2.3.12 static void PDB_HAL_SetIntCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the PDB interrupt.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

PDB HAL driver

<i>enable</i>	The switcher to assert the feature.
---------------	-------------------------------------

22.2.3.13 static void PDB_HAL_SetPreMultFactorMode (uint32_t *baseAddr*, pdb_mult_factor_mode_t *mode*) [inline], [static]

This function sets the PDB prescaler multiplication factor.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>mode</i>	Selection of mode, see to "pdb_mult_factor_mode_t".

22.2.3.14 static void PDB_HAL_SetContinuousModeCmd (uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function switches to enable the PDB continuous mode.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>enable</i>	The switcher to assert the feature.

22.2.3.15 static void PDB_HAL_SetLoadRegsCmd (uint32_t *baseAddr*) [inline], [static]

This function sets the LDOK bit and loads the delay registers value. Writing one to this bit updates the internal registers MOD, IDLY, CHnDLYm, DACINTx, and POyDLY with the values written to their buffers. The MOD, IDLY, CHnDLYm, DACINTx, and POyDLY take effect according to the load mode settings.

After one is written to the LDOK bit, the values in the buffers of above mentioned registers are not effective and cannot be written until the values in the buffers are loaded into their internal registers. The LDOK can be written only when the the PDB is enabled or as alone with it. It is automatically cleared either when the values in the buffers are loaded into the internal registers or when the PDB is disabled.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

22.2.3.16 static void PDB_HAL_SetModulusValue (*uint32_t baseAddr, uint32_t value*) [inline], [static]

This function sets the modulus value for the PDB module. When the counter reaches the setting value, it is automatically reset to zero. When in continuous mode, the counter begins to increase again.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>value</i>	The setting value of upper limit for PDB counter.

22.2.3.17 static uint32_t PDB_HAL_GetModulusValue (*uint32_t baseAddr*) [inline], [static]

This function gets the modulus value for the PDB module.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The current value of upper limit for counter.

22.2.3.18 static uint32_t PDB_HAL_GetCounterValue (*uint32_t baseAddr*) [inline], [static]

This function gets the PDB counter value.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The current counter value.

22.2.3.19 static void PDB_HAL_SetIntDelayValue (*uint32_t baseAddr, uint32_t value*) [inline], [static]

This function sets the interrupt delay milestone of the PDB counter. If enabled, a PDB interrupt is generated when the counter is equal to the setting value.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>value</i>	The setting value for interrupt delay milestone of PDB counter.

22.2.3.20 static uint32_t PDB_HAL_GetIntDelayValue (uint32_t *baseAddr*) [inline], [static]

This function gets the current interrupt delay milestone of the PDB counter.

Parameters

<i>baseAddr</i>	Register base address for the module.
-----------------	---------------------------------------

Returns

The current setting value for interrupt delay milestone of PDB counter.

22.2.3.21 void PDB_HAL_SetPreTriggerBackToBackCmd (uint32_t *baseAddr*, uint32_t *chn*, uint32_t *preChn*, bool *enable*)

This function switches to enable the pre-trigger back-to-back mode.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.
<i>enable</i>	Switcher to assert the feature.

22.2.3.22 void PDB_HAL_SetPreTriggerOutputCmd (uint32_t *baseAddr*, uint32_t *chn*, uint32_t *preChn*, bool *enable*)

This function switches to enable pre-trigger output.

Parameters

PDB HAL driver

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.
<i>enable</i>	Switcher to assert the feature.

22.2.3.23 void PDB_HAL_SetPreTriggerCmd (uint32_t *baseAddr*, uint32_t *chn*, uint32_t *preChn*, bool *enable*)

This function switches to enable the pre-trigger.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.
<i>enable</i>	Switcher to assert the feature.

22.2.3.24 static bool PDB_HAL_GetPreTriggerFlag (uint32_t *baseAddr*, uint32_t *chn*, uint32_t *preChn*) [inline], [static]

This function gets the flag which indicates the PDB counter has reached the pre-trigger delay value.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.

Returns

Flag status. True if the event is asserted.

22.2.3.25 void PDB_HAL_ClearPreTriggerFlag (uint32_t *baseAddr*, uint32_t *chn*, uint32_t *preChn*)

This function clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.

22.2.3.26 static bool PDB_HAL_GetPreTriggerSeqErrFlag (*uint32_t baseAddr, uint32_t chn, uint32_t preChn*) [inline], [static]

This function gets the flag which indicates whether a sequence error is detected.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.

Returns

Flag status. True if the event is asserted.

22.2.3.27 void PDB_HAL_ClearPreTriggerSeqErrFlag (*uint32_t baseAddr, uint32_t chn, uint32_t preChn*)

This function clears the flag which indicates that the sequence error has been detected.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.

22.2.3.28 void PDB_HAL_SetPreTriggerDelayCount (*uint32_t baseAddr, uint32_t chn, uint32_t preChn, uint32_t value*)

This function sets the pre-trigger delay value.

PDB HAL driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>preChn</i>	ADC channel group index for trigger.
<i>value</i>	Setting value for pre-trigger's delay value.

22.2.3.29 static void PDB_HAL_SetDacExtTriggerInputCmd (*uint32_t baseAddr, uint32_t dacChn, bool enable*) [inline], [static]

This function switches to enable the DAC external trigger input.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>dacChn</i>	DAC instance index for trigger.
<i>value</i>	Setting value for pre-trigger's delay value.
<i>enable</i>	Switcher to assert the feature.

22.2.3.30 static void PDB_HAL_SetDacIntervalTriggerCmd (*uint32_t baseAddr, uint32_t dacChn, bool enable*) [inline], [static]

This function switches to enable the DAC external trigger input.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>dacChn</i>	DAC instance index for trigger.
<i>enable</i>	Switcher to assert the feature.

22.2.3.31 static void PDB_HAL_SetDacIntervalValue (*uint32_t baseAddr, uint32_t dacChn, uint32_t value*) [inline], [static]

This function sets the interval value for the DAC trigger.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>dacChn</i>	DAC instance index for trigger.
<i>value</i>	Setting value for DAC trigger interval.

22.2.3.32 static uint32_t PDB_HAL_GetDacIntervalValue (uint32_t *baseAddr*, uint32_t *dacChn*) [inline], [static]

This function gets the interval value for the DAC trigger.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>dacChn</i>	DAC instance index for trigger.

Returns

The current setting value for DAC trigger interval.

22.2.3.33 void PDB_HAL_SetPulseOutCmd (uint32_t *baseAddr*, uint32_t *pulseChn*, bool *enable*)

This function switches to enable the pulse-out trigger.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>pulseChn</i>	Pulse-out channle index for trigger.
<i>enable</i>	Switcher to assert the feature.

22.2.3.34 static void PDB_HAL_SetPulseOutDelayForHigh (uint32_t *baseAddr*, uint32_t *pulseChn*, uint32_t *value*) [inline], [static]

This function sets the counter delay value for the pulse-out goes high.

PDB HAL driver

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>pulseChn</i>	Pulse-out channel index for trigger.
<i>value</i>	Setting value for PDB delay .

22.2.3.35 static void PDB_HAL_SetPulseOutDelayForLow (uint32_t *baseAddr*, uint32_t *pulseChn*, uint32_t *value*) [inline], [static]

This function sets the counter delay value for the pulse-out goes low.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>pulseChn</i>	Pulse-out channel index for trigger.
<i>value</i>	Setting value for PDB delay .

22.2.3.36 static void PDB_HAL_SetChnC1Reg (uint32_t *baseAddr*, uint32_t *chn*, uint32_t *regVal*) [inline], [static]

This function sets the value roughly into ChnC1 register for special time-critical case, like motor control, etc.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.
<i>regVal</i>	Setting value.

22.2.3.37 static uint32_t PDB_HAL_GetChnC1Reg (uint32_t *baseAddr*, uint32_t *chn*) [inline], [static]

This function gets the value roughly from ChnC1 register for special time-critical case, like motor control, etc.

Parameters

<i>baseAddr</i>	Register base address for the module.
<i>chn</i>	ADC instance index for trigger.

Returns

Setting value.

22.3 PDB Peripheral driver

This chapter describes the programming interface of the PDB Peripheral driver.

22.3.1 Overview

The PDB peripheral driver configures the PDB (Programmable Delay Block). It handles the triggers for ADC and DAC and pulse out to the CMP and the PDB counter.

22.3.2 PDB Driver model building

There is one main PDB counter for all triggers. When the indicated external trigger input arrives, the PDB counter launches and is increased by setting clock. The counter trigger milestones for ADC and DAC pulse out to the CMP and the PDB counter and wait for the PDB counter. Once the PDB counter hits each milestone, also called the critical delay value, the corresponding event is triggered and the trigger signal is sent out to trigger other peripherals. Therefore, the PDB module is a collector and manager of triggers.

22.3.3 PDB Initialization

The core feature of the PDB module is a programmable timer/counter. Additional features enable and set the milestone for the corresponding trigger. Therefore, the PDB module is first initialized as a programmable timer. To initialize the PDB driver, a configuration structure of the "pdb_user_config_t" type is required and should store an available configuration. The API of the [PDB_DRV_StructInitUserConfigForSoftTrigger\(\)](#) function provides a configuration which the PDB can use. However, this configuration is not sufficient for user-specific use cases. The user should provide a configuration suitable for the application requirements. Call the API of [PDB_DRV_Init\(\)](#) function to initialize the PDB timer/counter.

All triggers share the same counter. However, the DAC trigger does not share the same counting circle with other triggers. When the DAC's internal circle ends, the trigger is generated and the internal circle restarts.

For timing setting

The basic timing/counting step is set when initializing the main PDB counter:

The basic timing/counting step = $F_{BusClkH} / \text{pdb_user_config_t.clkPrescalerDivMode} / \text{pdb_user_config_t.multFactorMode}$

The $F_{BusClkH}$ is the frequency of bus clock in Hz. The "clkPrescalerDivMode" and "multFactorMode" are in the [pdb_user_config_t](#) structure. All triggering milestones are based on this step.

22.3.4 PDB Call diagram

Four kinds of typical use cases are designed for the PDB module.

- Normal Timer/Counter. Normal Timer/Counter is the basic case. The Timer/Counter starts after the PDB is initialized and the milestone for the PDB Timer/Counter is set. After it is triggered and when the counter hits the milestone, the interrupt request occurs if enabled. In continuous mode, when the counter hits the upper limitation, it returns zero and counts again.
- Trigger for ADC module. When the ADC trigger is enabled, a delay value for ADC trigger is set as the milestone. At least two ADC channel groups are provided. Likewise, there are more than two pre-triggers for ADC. Each pre-trigger is related to one channel group and can be enabled separately in the PDB module. When the PDB counter hits the milestone for the ADC pre-trigger, it triggers the ADC's conversion on the indicated channel group. To maximize the feature, the ADC should be configured to enable the hardware trigger mode.
- Trigger for the DAC module. A standalone DAC counter exists in the PDB module to trigger the DAC module. The user can set the upper limitation for the DAC counter. Once the counter reaches the upper limitation, it turns the DAC counter to zero and counts again. When the DAC counter hits the upper limitation, a DAC trigger is generated to trigger the DAC. This trigger updates the pointer of the DAC buffer. Although the DAC counter has its own setting for the upper limitation, it shares the same input trigger source, clock source, and reset control logic with the main PDB counter. When the PDB counter resets to zero, it forces the DAC counter to reset to zero. To maximize this feature, the DAC should be configured to enable the DAC buffer and hardware trigger mode.
- Trigger for pulse out to the CMP module. The pulse-out trigger is attached to the main PDB counter. There are two milestones for each pulse out channel, a milestone for level high and for level low, which makes a sample window for the CMP module.

These are the examples to initialize and configure the PDB driver for typical use cases.

Normal Timer/Counter:

```
// pdb_test_normal_timer.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_pdb_driver.h"
#include "fsl_os_abstraction.h"

#define PDB_TEST_TIMER_ROUNT_TIMES          (10U)

static volatile uint32_t MyPdbIntCounter = 0U;
static pdb_state_t MyPdbStateStruct;

extern void MyNop(void);
static void PDB_ISR_Counter(void);

void PDB_TEST_NormalTimer(uint32_t instance)
{
    pdb_user_config_t MyPdbUserConfigStruct;

    // Prepare the configuration structure. //
    PDB_DRV_StructInitUserConfigForSoftTrigger(&
        MyPdbUserConfigStruct);
    MyPdbUserConfigStruct.continuousModeEnable = true;
    MyPdbUserConfigStruct.delayValue = 0x002FU;
    MyPdbUserConfigStruct.pdbModulusValue = 0x004FU;

    // Initialize PDB counter. //
    PDB_DRV_Init(instance, &MyPdbUserConfigStruct, &MyPdbStateStruct);

    // Install the callback function. //
    PDB_SetCallback(instance, MyNop);
}
```

PDB Peripheral driver

```
PDB_DRV_InstallCallback(instance, PDB_ISR_Counter);

// Trigger the PDB. //
PDB_DRV_SoftTriggerCmd(instance);

// Wait for the counter. //
while (1U)
{
    if (MyPdbIntCounter >= PDB_TEST_TIMER_ROUND_TIMES)
    {
        printf("PDB counter goes %d rounds.\r\n", MyPdbIntCounter);
        break;
    }
    MyNop();
}

// De-initialize the PDB. //
PDB_DRV_Deinit(instance);

MyPdbIntCounter = 0U;
}

static void PDB_ISR_Counter(void)
{
    MyPdbIntCounter++;
}
```

Trigger for ADC module:

```
// pdb_test_adc_trigger.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_pdb_driver.h"
#include "fsl_adc_driver.h"
#include "fsl_os_abstraction.h"

#define PDB_TEST_ADC_TRIGGER_INSTANCE      (0U)
#define PDB_TEST_ADC_CHANNEL_GROUP        (1U)
#define PDB_TEST_ADC_CHANNEL_ID          (26U)
#define PDB_TEST_ADC_TRIGGER_TIMES       (10U)

static adc_state_t MyAdcState;
static pdb_state_t MyPdbStateStruct;
static volatile uint32_t MyAdcIntCounter = 0U;
static volatile uint32_t MyPdbIntCounter = 0U;

static void PDB_TEST_InitAdc(uint32_t instance, uint32_t chnGroup, uint32_t chn);
static void ADC_TEST_MyIsr(void);
static void PDB_ISR_Counter(void);
extern void MyNop(void);

void PDB_TEST_AdcTrigger(uint32_t instance)
{
    pdb_user_config_t MyPdbUserConfigStruct;
    pdb_adc_pre_trigger_config_t MyPdbAdcTriggerConfigStruct;
    uint32_t i;

    // Initialize the ADC that would be triggered. //
    PDB_TEST_InitAdc(PDB_TEST_ADC_TRIGGER_INSTANCE, PDB_TEST_ADC_CHANNEL_GROUP, PDB_TEST_ADC_CHANNEL_ID);

    // Prepare the configuration structure. //
    PDB_DRV_StructInitUserConfigForSoftTrigger(&
        MyPdbUserConfigStruct);
    MyPdbUserConfigStruct.continuousModeEnable = true;
```

```

MyPdbUserConfigStruct.delayValue = 0x002FU;
MyPdbUserConfigStruct.pdbModulusValue = 0x004FU;

// Initialize PDB counter. //
PDB_DRV_Init(instance, &MyPdbUserConfigStruct, &MyPdbStateStruct);

// Install the callback function. //
PDB_DRV_InstallCallback(instance, PDB_ISR_Counter);

// Initialize the ADC trigger in PDB. //
MyPdbAdcTriggerConfigStruct.backToBackModeEnable = false;
MyPdbAdcTriggerConfigStruct.triggerOutEnable = true;
MyPdbAdcTriggerConfigStruct.delayValue = 0x10U;
PDB_DRV_EnableAdcPreTrigger(instance, PDB_TEST_ADC_TRIGGER_INSTANCE,
    PDB_TEST_ADC_CHANNEL_GROUP, &MyPdbAdcTriggerConfigStruct);

// Trigger the PDB. //
PDB_DRV_SoftTriggerCmd(instance);

while (1U)
{
    if (MyAdcIntCounter >= PDB_TEST_ADC_TRIGGER_TIMES)
    {
        printf("ADC has been triggered by %d times.\r\n",
            MyAdcIntCounter);
        break;
    }

    MyNop();
}

PDB_DRV_DisableAdcPreTrigger(instance, PDB_TEST_ADC_TRIGGER_INSTANCE,
    PDB_TEST_ADC_CHANNEL_GROUP);
PDB_DRV_Deinit(instance);
ADC_DRV_Deinit(PDB_TEST_ADC_TRIGGER_INSTANCE);

MyAdcIntCounter = 0U;
}

static void PDB_TEST_InitAdc(uint32_t instance, uint32_t chnGroup, uint32_t chn)
{
    adc_calibration_param_t MyAdcCalibraitionParam;
    adc_user_config_t MyAdcUserConfig;
    adc_chn_config_t MyChnConfig;

    // Auto calibraion. //
    ADC_DRV_GetAutoCalibrationParam(instance, &MyAdcCalibraitionParam);
    ADC_DRV_SetCalibrationParam(instance, &MyAdcCalibraitionParam);

    // Initialization for interrupt mode but not continuous mode. //
    ADC_DRV_StructInitUserConfigForIntMode(&MyAdcUserConfig);
    MyAdcUserConfig.continuousConvEnable = false; // One conversion per-trigger. //
    MyAdcUserConfig.hwTriggerEnable = true; // Hardware trigger. //
    ADC_DRV_Init(instance, &MyAdcUserConfig, &MyAdcState);

    // Install Callback function into ISR. //
    ADC_DRV_InstallCallback(instance, chnGroup, ADC_TEST_MyIsr);

    // Set the channel. //
    MyChnConfig.chnNum = chn;
    MyChnConfig.diffEnable = false;
    MyChnConfig.intEnable = true;
    MyChnConfig.chnMux = kAdcChnMuxOfDefault;
    ADC_DRV_ConfigConvChn(instance, chnGroup, &MyChnConfig);
}

static void ADC_TEST_MyIsr(void)
{
}

```

PDB Peripheral driver

```
    MyAdcIntCounter++;
}

static void PDB_ISR_Counter(void)
{
    MyPdbIntCounter++;
}
```

Trigger for DAC module:

```
// pdb_test_dac_trigger.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <fsl_pdb_driver.h>
#include <fsl_dac_driver.h>
#include "fsl_os_abstraction.h"

#define PDB_TEST_TRIGGER_DAC_INSTANCE  (0U)
#define PDB_TEST_DAC_ARR_LEN          (16U)
#define PDB_TEST_DAC_TRIGGER_TIMES   (10U)

uint16_t MyPdbDemoDacBuff[PDB_TEST_DAC_ARR_LEN] =
{
    0U,      256U,    512U,    768U,
    1024U,   1280U,   1536U,   1792U,
    2048U,   2304U,   2560U,   2816U,
    3072U,   3328U,   3584U,   3840U
};

static dac_state_t MyDacStateStructForBufferFIFO;
static pdb_state_t MyPdbStateStruct;
static volatile uint32_t MyPdbIntCounter = 0U;
static volatile uint32_t MyDacIntCounter = 0U;

extern void MyNop(void);
static void DAC_ISR_Buffer(void);
static void PDB_TEST_InitDac(uint32_t instance);
static void PDB_ISR_Counter(void);

void PDB_TEST_DacTrigger(uint32_t instance)
{
    pdb_user_config_t MyPdbUserConfigStruct;
    pdb_dac_trigger_config_t MyPdbDacTriggerConfigStruct;
    uint32_t i;

    // Initialize the DAC module. //
    PDB_TEST_InitDac(PDB_TEST_TRIGGER_DAC_INSTANCE);

    // Prepare the configuration structure. //
    PDB_DRV_StructInitUserConfigForSoftTrigger(&
        MyPdbUserConfigStruct);
    MyPdbUserConfigStruct.continuousModeEnable = true;
    MyPdbUserConfigStruct.delayValue = 0x002FU;
    MyPdbUserConfigStruct.pdbModulusValue = 0x004FU;

    // Initialize PDB counter. //
    PDB_DRV_Init(instance, &MyPdbUserConfigStruct, &MyPdbStateStruct);

    // Install the PDB ISR. //
    PDB_DRV_InstallCallback(instance, PDB_ISR_Counter);

    // Initialize the DAC trigger. //
    MyPdbDacTriggerConfigStruct.extTriggerInputEnable = false;
    MyPdbDacTriggerConfigStruct.intervalValue = 4U;
```

```

PDB_DRV_EnableDacTrigger(instance, PDB_TEST_TRIGGER_DAC_INSTANCE,
    &MyPdbDacTriggerConfigStruct);

// Trigger the PDB. //
PDB_DRV_SoftTriggerCmd(instance);

while (1)
{
    if (MyPdbIntCounter >= PDB_TEST_DAC_TRIGGER_TIMES)
    {
        printf("DAC buffer interrupt Start/Upper/Watermark counter: %d.\r\n",
            MyDacIntCounter);
        break;
    }

    MyNop();
}

PDB_DRV_DisableDacTrigger(instance, PDB_TEST_TRIGGER_DAC_INSTANCE);
PDB_DRV_Deinit(instance);
DAC_DRV_Deinit(PDB_TEST_TRIGGER_DAC_INSTANCE);
MyDacIntCounter = 0U;
MyPdbIntCounter = 0U;
}

static void PDB_TEST_InitDac(uint32_t instance)
{
    dac_user_config_t MyDacUserConfigStruct;
    dac_buff_config_t MyDacBuffConfigStruct;

    // Initialize the DAC converter. //
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);
    MyDacUserConfigStruct.triggerMode = kDacTriggerByHardware; // Hardware
        trigger. //

    DAC_DRV_Init(instance, &MyDacUserConfigStruct);

    // Enable buffer. //
    // Enable the feature of DAC internal buffer. //
    MyDacBuffConfigStruct.bufIndexWatermarkIntEnable = true;
    MyDacBuffConfigStruct.bufIndexStartIntEnable = true;
    MyDacBuffConfigStruct.bufIndexUpperIntEnable = true;
    MyDacBuffConfigStruct.dmaEnable = false;
    MyDacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word
    ;
    MyDacBuffConfigStruct.bufWorkMode = kDacBuffWorkAsFIFOMode;
    MyDacBuffConfigStruct.bufUpperIndex = PDB_TEST_DAC_ARR_LEN - 1;
    DAC_DRV_EnableBuff(instance, &MyDacBuffConfigStruct, &MyDacStateStructForBufferFIFO);

    // Fill the buffer with setting data. //
    DAC_DRV_SetBuffValue(instance, 0U, PDB_TEST_DAC_ARR_LEN, MyPdbDemoDacBuff);
    // Register the callback function for DAC buffer event. //
    DAC_DRV_InstallCallback(instance, DAC_ISR_Buffer);
}

static void DAC_ISR_Buffer(void)
{
    MyDacIntCounter++;
}

static void PDB_ISR_Counter(void)
{
    MyPdbIntCounter++;
}

```

Trigger for Pulse-out:

PDB Peripheral driver

```
// pdb_test_pulse_out_trigger.c //

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "fsl_pdb_driver.h"
#include "fsl_os_abstraction.h"

#define PDB_TEST_PULSE_OUT_CHANNEL_ID      (0U)
#define PDB_TEST_PULSE_OUT_TRIGGER_TIMES    (10U)

static pdb_state_t MyPdbStateStruct;
static volatile uint32_t MyPdbIntCounter = 0U;
extern void MyNop(void);
static void PDB_ISR_Counter(void);

void PDB_TEST_PulseOutTrigger(uint32_t instance)
{
    pdb_user_config_t MyPdbUserConfigStruct;
    pdb_pulse_out_trigger_config_t MyPulseOutTriggerConfigStruct;
    uint32_t i, j;

    // Prepare the configuration structure. //
    PDB_DRV_StructInitUserConfigForSoftTrigger(&
        MyPdbUserConfigStruct);
    MyPdbUserConfigStruct.continuousModeEnable = true;
    MyPdbUserConfigStruct.delayValue = 0x002FU;
    MyPdbUserConfigStruct.pdbModulusValue = 0x004FU;

    // Initialize PDB counter. //
    PDB_DRV_Init(instance, &MyPdbUserConfigStruct, &MyPdbStateStruct);

    // Install the callback function. //
    PDB_DRV_InstallCallback(instance, PDB_ISR_Counter);

    // Initialize the Pulse out in PDB. //
    MyPulseOutTriggerConfigStruct.pulseOutHighValue = 0x0010U;
    MyPulseOutTriggerConfigStruct.pulseOutLowValue = 0x0020U;
    PDB_DRV_EnablePulseOutTrigger(instance, PDB_TEST_PULSE_OUT_CHANNEL_ID, &
        MyPulseOutTriggerConfigStruct);

    // Trigger the PDB. //
    PDB_DRV_SoftTriggerCmd(instance);

    while (1U)
    {
        if (MyPdbIntCounter >= PDB_TEST_PULSE_OUT_TRIGGER_TIMES)
        {
            printf("Pulse-out has been triggered by %d times.\r\n",
                MyPdbIntCounter);
            break;
        }
        MyNop();
    }

    PDB_DRV_DisablePulseOutTrigger(instance, PDB_TEST_PULSE_OUT_CHANNEL_ID);
    PDB_DRV_Deinit(instance);
    MyPdbIntCounter = 0U;
}

static void PDB_ISR_Counter(void)
{
    MyPdbIntCounter++;
}
```

Data Structures

- struct `pdb_user_config_t`
Defines the structure to configure the PDB counter. [More...](#)
- struct `pdb_adc_pre_trigger_config_t`
Defines the structure to configure the ADC pre-trigger in the PDB module. [More...](#)
- struct `pdb_dac_trigger_config_t`
Defines the structure to configuring the DAC trigger in the PDB module. [More...](#)
- struct `pdb_pulse_out_trigger_config_t`
Defines the structure to configure the pulse-out trigger in the PDB module. [More...](#)

Enumerations

- enum `pdb_adc_pre_trigger_flag_t` {

`kPdbAdcPreChnFlag` = 0U,
`kPdbAdcPreChnErrFlag` = 1U }
- Defines the type of flag for PDB pre-trigger events.*

Functions

- `pdb_status_t PDB_DRV_StructInitUserConfigForSoftTrigger (pdb_user_config_t *userConfigPtr)`
Populates the initial user configuration for software trigger mode.
- `pdb_status_t PDB_DRV_Init (uint32_t instance, pdb_user_config_t *userConfigPtr)`
Initializes the PDB counter and trigger input.
- void `PDB_DRV_Deinit (uint32_t instance)`
De-initializes the PDB module.
- void `PDB_DRV_SoftTriggerCmd (uint32_t instance)`
Triggers the PDB with a software trigger.
- `uint32_t PDB_DRV_GetCurrentCounter (uint32_t instance)`
Gets the current counter value in the PDB module.
- `bool PDB_DRV_GetPdbCounterIntFlag (uint32_t instance)`
Gets the PDB interrupt flag.
- void `PDB_DRV_ClearPdbCounterIntFlag (uint32_t instance)`
Clears the interrupt flag.
- `pdb_status_t PDB_DRV_EnableAdcPreTrigger (uint32_t instance, uint32_t adcChn, uint32_t preChn, pdb_adc_pre_trigger_config_t *adcPreTriggerConfigPtr)`
Enables the ADC pre-trigger with its configuration.
- void `PDB_DRV_DisableAdcPreTrigger (uint32_t instance, uint32_t adcChn, uint32_t preChn)`
Disables the ADC pre-trigger.
- `bool PDB_DRV_GetAdcPreTriggerFlag (uint32_t instance, uint32_t adcChn, uint32_t preChn, pdb_adc_pre_trigger_flag_t flag)`
Gets the ADC pre-trigger flag.
- void `PDB_DRV_ClearAdcPreTriggerFlag (uint32_t instance, uint32_t adcChn, uint32_t preChn, pdb_adc_pre_trigger_flag_t flag)`
Clears the ADC pre-trigger flag.
- `pdb_status_t PDB_DRV_EnableDacTrigger (uint32_t instance, uint32_t dacChn, pdb_dac_trigger_config_t *dacTriggerConfigPtr)`

PDB Peripheral driver

- void **PDB_DRV_DisableDacTrigger** (uint32_t instance, uint32_t dacChn)
Disables the DAC trigger.
- **pdb_status_t PDB_DRV_EnablePulseOutTrigger** (uint32_t instance, uint32_t pulseChn, **pdb_pulse_out_trigger_config_t** *pulseOutTriggerConfigPtr)
Enables the pulse-out trigger with its configuration.
- void **PDB_DRV_DisablePulseOutTrigger** (uint32_t instance, uint32_t pulseChn)
Disables the pulse-out trigger.

Variables

- const uint32_t **g_pdbBaseAddr** []
Table of base addresses for PDB instances.
- const IRQn_Type **g_pdbIrqId** [HW_PDB_INSTANCE_COUNT]
Table to save PDB IRQ enum numbers defined in CMSIS header file.

22.3.5 Data Structure Documentation

22.3.5.1 struct **pdb_user_config_t**

This structure keeps the configuration for the PDB basic counter. The PDB counter is the core part of the PDB module. When initialized, the PDB module can act as a simple counter.

Data Fields

- **pdb_load_mode_t loadMode**
Selects the mode to load timing registers after set load operation.
- bool **seqErrIntEnable**
Switch to enable the PDB sequence error interrupt.
- bool **dmaEnable**
Switch to enable DMA support for PDB instead of interrupt.
- **pdb_clk_prescaler_div_mode_t clkPrescalerDivMode**
Select the prescaler that counting uses the peripheral clock divided by multiplication factor.
- **pdb_trigger_src_mode_t triggerSrcMode**
Select the input source of trigger.
- bool **intEnable**
Switch to enable the PDB interrupt for counter reaches to the modulus.
- **pdb_mult_factor_mode_t multFactorMode**
Select the multiplication factor for prescalar.
- bool **continuousModeEnable**
Switch to enable the continuous mode.
- uint32_t **pdbModulusValue**
Set the value for PDB counter's modulus.
- uint32_t **delayValue**
Set the value for PDB counter to cause PDB interrupt.

22.3.5.1.0.47 Field Documentation

- 22.3.5.1.0.47.1 `pdb_load_mode_t pdb_user_config_t::loadMode`
- 22.3.5.1.0.47.2 `bool pdb_user_config_t::seqErrIntEnable`
- 22.3.5.1.0.47.3 `bool pdb_user_config_t::dmaEnable`
- 22.3.5.1.0.47.4 `pdb_clk_prescaler_div_mode_t pdb_user_config_t::clkPrescalerDivMode`
- 22.3.5.1.0.47.5 `pdb_trigger_src_mode_t pdb_user_config_t::triggerSrcMode`
- 22.3.5.1.0.47.6 `bool pdb_user_config_t::intEnable`
- 22.3.5.1.0.47.7 `pdb_mult_factor_mode_t pdb_user_config_t::multFactorMode`
- 22.3.5.1.0.47.8 `bool pdb_user_config_t::continuousModeEnable`
- 22.3.5.1.0.47.9 `uint32_t pdb_user_config_t::pdbModulusValue`
- 22.3.5.1.0.47.10 `uint32_t pdb_user_config_t::delayValue`

22.3.5.2 struct `pdb_adc_pre_trigger_config_t`

This structure keeps the configuration for ADC pre-trigger in PDB module.

Data Fields

- `bool backToBackModeEnable`
Switch to enable the back-to-back mode.
- `bool preTriggerOutEnable`
Switch to enable trigger out the pre-channel.
- `uint32_t delayValue`
Set the value for ADC pre-trigger's delay value.

22.3.5.2.0.48 Field Documentation

- 22.3.5.2.0.48.1 `bool pdb_adc_pre_trigger_config_t::backToBackModeEnable`
- 22.3.5.2.0.48.2 `bool pdb_adc_pre_trigger_config_t::preTriggerOutEnable`
- 22.3.5.2.0.48.3 `uint32_t pdb_adc_pre_trigger_config_t::delayValue`

22.3.5.3 struct `pdb_dac_trigger_config_t`

This structure keeps the configuration for DAC trigger in the PDB module.

Data Fields

- bool `extTriggerInputEnable`
Switch to enable the external trigger for DAC interval counter.

22.3.5.3.0.49 Field Documentation

22.3.5.3.0.49.1 bool `pdb_dac_trigger_config_t::extTriggerInputEnable`

22.3.5.4 struct `pdb_pulse_out_trigger_config_t`

This structure keeps the configuration for the pulse-out trigger in the PDB module.

Data Fields

- uint32_t `pulseOutHighValue`
Set the value for that pulse out goes high when the PDB counter reaches to this value.
- uint32_t `pulseOutLowValue`
Set the value for that pulse out goes low when the PDB counter reaches to this value.

22.3.5.4.0.50 Field Documentation

22.3.5.4.0.50.1 uint32_t `pdb_pulse_out_trigger_config_t::pulseOutHighValue`

22.3.5.4.0.50.2 uint32_t `pdb_pulse_out_trigger_config_t::pulseOutLowValue`

22.3.6 Enumeration Type Documentation

22.3.6.1 enum `pdb_adc_pre_trigger_flag_t`

Enumerator

`kPdbAdcPreChnFlag` ADC pre-trigger flag when the counter reaches to pre-trigger's delay value.

`kPdbAdcPreChnErrFlag` ADC pre-trigger sequence error flag.

22.3.7 Function Documentation

22.3.7.1 `pdb_status_t PDB_DRV_StructInitUserConfigForSoftTrigger (pdb_user_config_t * userConfigPtr)`

This function populates the initial user configuration. It is set as one-time, software trigger mode. The PDB modulus and delay value are all set to maximum. The PDB interrupt is enabled and causes the PDB interrupt when the counter hits the delay value. Then, the PDB module is initialized as a normal timer if no other setting is changed. The settings are:

.loadMode = kPdbLoadImmediately; .seqErrIntEnable = false; .dmaEnable = false; .clkPrescalerDivMode = kPdbClkPreDivBy128; .triggerSrcMode = kPdbSoftTrigger; .intEnable = true; .multFactorMode = k-

PdbMultFactorAs40; .continuousModeEnable = false; .pdbModulusValue = 0xFFFFU; .delayValue = 0xFFFFU;

Parameters

<i>userConfigPtr</i>	Pointer to the user configuration structure. See the "pdb_user_config_t".
----------------------	---

Returns

Execution status.

22.3.7.2 `pdb_status_t PDB_DRV_Init (uint32_t instance, pdb_user_config_t * userConfigPtr)`

This function initializes the PDB counter and triggers the input. It resets PDB registers and enables the PDB clock. Therefore, it should be called before any other operation. After it is initialized, the PDB can act as a triggered timer, which enables other features in PDB module.

Parameters

<i>instance</i>	PDB instance ID.
<i>userConfigPtr</i>	Pointer to the user configuration structure. See the "pdb_user_config_t".

Returns

Execution status.

22.3.7.3 `void PDB_DRV_Deinit (uint32_t instance)`

This function de-initializes the PDB module. Calling this function shuts down the PDB module and reduces the power consumption.

Parameters

<i>instance</i>	PDB instance ID.
-----------------	------------------

22.3.7.4 `void PDB_DRV_SoftTriggerCmd (uint32_t instance)`

This function triggers the PDB with a software trigger. When the PDB is set to use the software trigger as input, calling this function triggers the PDB.

PDB Peripheral driver

Parameters

<i>instance</i>	PDB instance ID.
-----------------	------------------

22.3.7.5 uint32_t PDB_DRV_GetCurrentCounter (uint32_t *instance*)

This function gets the current counter value.

Parameters

<i>instance</i>	PDB instance ID.
-----------------	------------------

Returns

Current PDB counter value.

22.3.7.6 bool PDB_DRV_GetPdbCounterIntFlag (uint32_t *instance*)

This function gets the PDB interrupt flag. It is asserted if the PDB interrupt occurs.

Parameters

<i>instance</i>	PDB instance ID.
-----------------	------------------

Returns

Assertion of indicated event.

22.3.7.7 void PDB_DRV_ClearPdbCounterIntFlag (uint32_t *instance*)

This function clears the interrupt flag.

Parameters

<i>instance</i>	PDB instance ID.
-----------------	------------------

22.3.7.8 pdb_status_t PDB_DRV_EnableAdcPreTrigger (uint32_t *instance*, uint32_t *adcChn*, uint32_t *preChn*, pdb_adc_pre_trigger_config_t * *adcPreTriggerConfigPtr*)

This function enables the ADC pre-trigger with its configuration. The setting value takes effect according to the load-mode configured during the PDB initialization, although the load operation was done inside the function. This is an additional function based on the PDB counter.

Parameters

<i>instance</i>	PDB instance ID.
<i>adcChn</i>	ADC trigger channel, related to the ADC instance.
<i>preChn</i>	ADC pre-trigger channel, related to the ADC channel group instance.
<i>adcPreTriggerConfigPtr</i>	Pointer to structure of configuration, see to "pdb_adc_pre_trigger_config_t".

Returns

Execution status.

22.3.7.9 void PDB_DRV_DisableAdcPreTrigger (uint32_t *instance*, uint32_t *adcChn*, uint32_t *preChn*)

This function disables the ADC pre-trigger. The PDB works the same way as when the pre-trigger was not enabled.

Parameters

<i>instance</i>	PDB instance ID.
<i>adcChn</i>	ADC trigger channel, related to the ADC instance.
<i>preChn</i>	ADC pre-trigger channel, related to the ADC channel group instance.

22.3.7.10 bool PDB_DRV_GetAdcPreTriggerFlag (uint32_t *instance*, uint32_t *adcChn*, uint32_t *preChn*, pdb_adc_pre_trigger_flag_t *flag*)

This function gets the pre-trigger flag for the PDB module. It is asserted if a related event occurs.

Parameters

<i>instance</i>	PDB instance ID.
<i>adcChn</i>	ADC trigger channel, related to the ADC instance.
<i>preChn</i>	ADC pre-trigger channel, related to the ADC channel group instance.
<i>flag</i>	Indicated flag for event.

Returns

Assertion of indicated event.

PDB Peripheral driver

**22.3.7.11 void PDB_DRV_ClearAdcPreTriggerFlag (uint32_t *instance*, uint32_t *adcChn*,
 uint32_t *preChn*, pdb_adc_pre_trigger_flag_t *flag*)**

This function clears the ADC pre-trigger flag for the PDB module.

Parameters

<i>instance</i>	PDB instance ID.
<i>adcChn</i>	ADC trigger channel, related to the ADC instance.
<i>preChn</i>	ADC pre-trigger channel, related to the ADC channel group instance.
<i>flag</i>	Indicated flag for event.

22.3.7.12 **pdb_status_t PDB_DRV_EnableDacTrigger (uint32_t *instance*, uint32_t *dacChn*, pdb_dac_trigger_config_t * *dacTriggerConfigPtr*)**

This function enables the DAC trigger with its configuration. The setting value takes effect according to the load mode configured during PDB initialization, although the load operation is done inside the function. This is an additional function based on the PDB counter.

Parameters

<i>instance</i>	PDB instance ID.
<i>dacChn</i>	DAC trigger channel, related to the DAC instance.
<i>dacTriggerConfigPtr</i>	Pointer to structure of configuration, see to "pdb_dac_trigger_config_t".

Returns

Execution status.

22.3.7.13 **void PDB_DRV_DisableDacTrigger (uint32_t *instance*, uint32_t *dacChn*)**

This function disables the DAC trigger. The PDB works the same as when the DAC trigger was not enabled.

Parameters

<i>instance</i>	PDB instance ID.
<i>dacChn</i>	DAC trigger channel, related to the DAC instance.

22.3.7.14 **pdb_status_t PDB_DRV_EnablePulseOutTrigger (uint32_t *instance*, uint32_t *pulseChn*, pdb_pulse_out_trigger_config_t * *pulseOutTriggerConfigPtr*)**

This function enables the pulse-out trigger with its configuration. The setting value takes effect according to the load mode configured during the PDB initialization, although the load operation is done inside the function. This is an additional function based on the PDB counter.

PDB Peripheral driver

Parameters

<i>instance</i>	PDB instance ID.
<i>pulseChn</i>	Pulse out trigger channel.
<i>pulseOutTriggerConfigPtr</i>	Pointer to structure of configuration, see to "pdb_pulse_out_trigger_config_t".

Returns

Execution status.

22.3.7.15 void PDB_DRV_DisablePulseOutTrigger (uint32_t *instance*, uint32_t *pulseChn*)

This function disables the pulse-out trigger. The PDB works the same as when the pulse out-trigger was not been enabled.

Parameters

<i>instance</i>	PDB instance ID.
<i>pulseChn</i>	Pulse out trigger channel.

22.3.8 Variable Documentation

22.3.8.1 const uint32_t g_pdbBaseAddr[]

22.3.8.2 const IRQn_Type g_pdbIrqId[HW_PDB_INSTANCE_COUNT]

Chapter 23

Periodic Interrupt Timer (PIT)

23.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Periodic Interrupt Timer (PIT) block of Kinetis devices.

Modules

- [PIT HAL driver](#)
- [PIT Peripheral driver](#)

PIT HAL driver

23.2 PIT HAL driver

23.2.1 Overview

This section describes the programming interface of the PIT HAL driver.

Initialization

- static void **PIT_HAL_Enable** (uint32_t baseAddr)
Enables the PIT module.
- static void **PIT_HAL_Disable** (uint32_t baseAddr)
Disables the PIT module.
- static void **PIT_HAL_SetTimerRunInDebugCmd** (uint32_t baseAddr, bool timerRun)
Configures the timers to continue running or to stop in debug mode.

Timer Start and Stop

- static void **PIT_HAL_StartTimer** (uint32_t baseAddr, uint32_t channel)
Starts the timer counting.
- static void **PIT_HAL_StopTimer** (uint32_t baseAddr, uint32_t channel)
Stops the timer from counting.
- static bool **PIT_HAL_IsTimerRunning** (uint32_t baseAddr, uint32_t channel)
Checks to see whether the current timer is started or not.

Timer Period

- static void **PIT_HAL_SetTimerPeriodByCount** (uint32_t baseAddr, uint32_t channel, uint32_t count)
Sets the timer period in units of count.
- static uint32_t **PIT_HAL_GetTimerPeriodByCount** (uint32_t baseAddr, uint32_t channel)
Returns the current timer period in units of count.
- static uint32_t **PIT_HAL_ReadTimerCount** (uint32_t baseAddr, uint32_t channel)
Reads the current timer counting value.

Interrupt

- static void **PIT_HAL_SetIntCmd** (uint32_t baseAddr, uint32_t channel, bool enable)
Enables or disables the timer interrupt.
- static bool **PIT_HAL.GetIntCmd** (uint32_t baseAddr, uint32_t channel)
Checks whether the timer interrupt is enabled or not.
- static void **PIT_HAL_ClearIntFlag** (uint32_t baseAddr, uint32_t channel)
Clears the timer interrupt flag.
- static bool **PIT_HAL_IsIntPending** (uint32_t baseAddr, uint32_t channel)
Reads the current timer timeout flag.

23.2.2 Function Documentation

23.2.2.1 static void PIT_HAL_Enable(uint32_t *baseAddr*) [inline], [static]

This function enables the PIT timer clock (Note: this function does not un-gate the system clock gating control). It should be called before any other timer related setup.

PIT HAL driver

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
-----------------	--

23.2.2.2 static void PIT_HAL_Disable(uint32_t *baseAddr*) [inline], [static]

This function disables all PIT timer clocks (Note: it does not affect the SIM clock gating control).

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
-----------------	--

23.2.2.3 static void PIT_HAL_SetTimerRunInDebugCmd(uint32_t *baseAddr*, bool *timerRun*) [inline], [static]

In debug mode, the timers may or may not be frozen, based on the configuration of this function. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (for example, the timer values), and continue the operation.

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>timerRun</i>	Timers run or stop in debug mode. <ul style="list-style-type: none">• true: Timers continue to run in debug mode.• false: Timers stop in debug mode.

23.2.2.4 static void PIT_HAL_StartTimer(uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

After calling this function, timers load the start value as specified by the function [PIT_HAL_SetTimerPeriodByCount\(uint32_t baseAddr, uint32_t channel, uint32_t count\)](#), count down to 0, and load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the time-out interrupt flag.

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

23.2.2.5 static void PIT_HAL_StopTimer (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

This function stops every timer from counting. Timers reload their periods respectively after they call the PIT_HAL_StartTimer the next time.

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

23.2.2.6 static bool PIT_HAL_IsTimerRunning (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

Returns

Current timer running status -true: Current timer is running. -false: Current timer has stopped.

23.2.2.7 static void PIT_HAL_SetTimerPeriodByCount (*uint32_t baseAddr, uint32_t channel, uint32_t count*) [inline], [static]

Timers begin counting from the value set by this function. The counter period of a running timer can be modified by first stopping the timer, setting a new load value, and starting the timer again. If timers are not restarted, the new value is loaded after the next trigger event.

Parameters

PIT HAL driver

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number
<i>count</i>	Timer period in units of count

23.2.2.8 static uint32_t PIT_HAL_GetTimerPeriodByCount (uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

Returns

Timer period in units of count

23.2.2.9 static uint32_t PIT_HAL_ReadTimerCount (uint32_t *baseAddr*, uint32_t *channel*) [inline], [static]

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

Returns

Current timer counting value

23.2.2.10 static void PIT_HAL_SetIntCmd (uint32_t *baseAddr*, uint32_t *channel*, bool *enable*) [inline], [static]

If enabled, an interrupt happens when a timeout event occurs (Note: NVIC should be called to enable pit interrupt in system level).

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number
<i>enable</i>	Enable or disable interrupt. <ul style="list-style-type: none"> • true: Generate interrupt when timer counts to 0. • false: No interrupt is generated.

23.2.2.11 static bool PIT_HAL_GetIntCmd (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

Returns

Status of enabled or disabled interrupt

- true: Interrupt is enabled.
- false: Interrupt is disabled.

23.2.2.12 static void PIT_HAL_ClearIntFlag (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

This function clears the timer interrupt flag after a timeout event occurs.

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

23.2.2.13 static bool PIT_HAL_IsIntPending (*uint32_t baseAddr, uint32_t channel*) [inline], [static]

Every time the timer counts to 0, this flag is set.

PIT HAL driver

Parameters

<i>baseAddr</i>	Base address for current PIT instance.
<i>channel</i>	Timer channel number

Returns

Current status of the timeout flag

- true: Timeout has occurred.
- false: Timeout has not yet occurred.

23.3 PIT Peripheral driver

23.3.1 Overview

This section describes the programming interface of the PIT Peripheral driver. The PIT driver configures PIT timers and initializes and configures timer periods.

23.3.2 PIT Initialization

1. To initialize the PIT module, call the `PIT_DRV_Init` function. This function enables the PIT module and clock automatically.
2. The parameter passed in the `PIT_DRV_Init` configures the timers to run or stop in debug mode. To use one timer channel, call the `PIT_DRV_InitChannel` initialize that channel.

This is example code to initialize and configure the driver:

```
// Define device configuration.
const pit_config_t pitInit = {
    .isInterruptEnabled = false, // Disable timer interrupt.
    .isTimerChained = false,    // Meaningless for timer 0.
    .periodUs = 20U            // Set timer period to 20 us.
};

// Initialize PIT instance 0. Timers will stop running in debug mode.
PIT_DRV_Init(0, stop);

// Initialize PIT instance 0, timer 0.
PIT_DRV_InitChannel(0, 0, &pitInit);
```

23.3.3 PIT Timer Period

The PIT driver provides four ways to set the timer period.

1. The `PIT_DRV_InitChannel` function sets the timer period in microseconds. It is only applicable when initializing the channel.
2. The void `PIT_DRV_SetTimerPeriodByUs(uint32_t instance, uint32_t timer, uint32_t us)` function sets the timer period in microseconds. It is applicable at any time.
3. The void `PIT_DRV_SetLifetimeTimerPeriodByUs(uint32_t instance, uint64_t us)` function sets the lifetime timer period in microseconds. It only supports specific MCUs. Check the appropriate reference manual before using this function.
4. The void `PIT_HAL_SetTimerPeriodByCount(uint32_t instance, uint32_t timer, uint32_t count)` function sets the timer period in units of count. To use this function, include the `fsl_pit_hal.h`.

To read the current timer counting value in microseconds, call the `uint32_t PIT_DRV_ReadTimerUs(uint32_t instance, uint32_t timer)` function.

PIT Peripheral driver

23.3.4 PIT Timer Operation

After the timer setting is complete, call the void `PIT_DRV_StartTimer(uint32_t timer)` function to start timer counting. Call the void `PIT_DRV_StopTimer(uint32_t instance, uint32_t timer)` function to stop it at any time.

If you want to close the PIT module entirely, call the void `PIT_DRV_Deinit(uint32_t instance)` function. This disables both the PIT module and the clock gate.

Data Structures

- struct `pit_user_config_t`
PIT timer configuration structure. [More...](#)

Typedefs

- typedef void(* `pit_isr_callback_t`)`(uint32_t channel)`
PIT ISR callback function typedef.

Variables

- const uint32_t `g_pitBaseAddr []`
Table of base addresses for pit instances.

Initialize and Shutdown

- void `PIT_DRV_Init` (`uint32_t instance, bool isRunInDebug`)
Initializes the PIT module.
- void `PIT_DRV_Deinit` (`uint32_t instance`)
Disables the PIT module and gate control.
- void `PIT_DRV_InitChannel` (`uint32_t instance, uint32_t channel, const pit_user_config_t *config`)
Initializes the PIT channel.

Timer Start and Stop

- void `PIT_DRV_StartTimer` (`uint32_t instance, uint32_t channel`)
Starts the timer counting.
- void `PIT_DRV_StopTimer` (`uint32_t instance, uint32_t channel`)
Stops the timer counting.

Timer Period

- void `PIT_DRV_SetTimerPeriodByUs` (`uint32_t instance, uint32_t channel, uint32_t us`)

- `uint32_t PIT_DRV_GetTimerPeriodByUs` (`uint32_t instance, uint32_t channel`)
 - Sets the timer period in microseconds.*
 - Gets the timer period in microseconds for one single channel.*
- `uint32_t PIT_DRV_ReadTimerUs` (`uint32_t instance, uint32_t channel`)
 - Reads the current timer value in microseconds.*
- `void PIT_DRV_SetTimerPeriodByCount` (`uint32_t instance, uint32_t channel, uint32_t count`)
 - Sets the timer period in units of count.*
- `uint32_t PIT_DRV_GetTimerPeriodByCount` (`uint32_t instance, uint32_t channel`)
 - Returns the current timer period in units of count.*
- `uint32_t PIT_DRV_ReadTimerCount` (`uint32_t instance, uint32_t channel`)
 - Reads the current timer counting value.*

ISR Callback Function

- `pit_isr_callback_t PIT_DRV_InstallCallback` (`uint32_t instance, uint32_t channel, pit_isr_callback_t function`)
 - Registers the PIT ISR callback function.*

23.3.5 Data Structure Documentation

23.3.5.1 struct pit_user_config_t

Defines a structure PitConfig and uses the `PIT_DRV_InitChannel()` function to make necessary initializations. You may also use the remaining functions for PIT configuration.

Note

The timer chain feature is not valid in all devices. Check the `fsl_pit_features.h` for accurate settings. If it's not valid, the value set here will be bypassed inside the `PIT_DRV_InitChannel()` function.

Data Fields

- `bool isInterruptEnabled`
 - Timer interrupt 0-disable/1-enable.*
- `bool isTimerChained`
 - Chained with previous timer, 0-not/1-chained.*
- `uint32_t periodUs`
 - Timer period in unit of microseconds.*

23.3.6 Function Documentation

23.3.6.1 void PIT_DRV_Init (`uint32_t instance, bool isRunInDebug`)

This function must be called before calling all the other PIT driver functions. This function ungates the PIT clock and enables the PIT module. The `isRunInDebug` passed into function affects all timer channels.

PIT Peripheral driver

Parameters

<i>instance</i>	PIT module instance number.
<i>isRunInDebug</i>	Timers run or stop in debug mode. <ul style="list-style-type: none">• true: Timers continue to run in debug mode.• false: Timers stop in debug mode.

23.3.6.2 void PIT_DRV_Deinit(uint32_t *instance*)

This function disables all PIT interrupts and PIT clock. It then gates the PIT clock control. PIT_DRV_Init must be called if you want to use PIT again.

Parameters

<i>instance</i>	PIT module instance number.
-----------------	-----------------------------

23.3.6.3 void PIT_DRV_InitChannel(uint32_t *instance*, uint32_t *channel*, const pit_user_config_t * *config*)

This function initializes the PIT timers by using a channel. Pass in the timer number and its configuration structure. Timers do not start counting by default after calling this function. The function PIT_DRV_StartTimer must be called to start the timer counting. Call the PIT_DRV_SetTimerPeriodByUs to re-set the period.

This is an example demonstrating how to define a PIT channel configuration structure:

```
pit_user_config_t pitTestInit = {
    .isInterruptEnabled = true,
    // Only takes effect when chain feature is available.
    // Otherwise, pass in arbitrary value(true/false).
    .isTimerChained = false,
    // In unit of microseconds.
    .periodUs = 1000,
};
```

Parameters

<i>instance</i>	PIT module instance number.
-----------------	-----------------------------

<i>channel</i>	Timer channel number.
<i>config</i>	PIT channel configuration structure.

23.3.6.4 void PIT_DRV_StartTimer (*uint32_t instance, uint32_t channel*)

After calling this function, timers load period value, count down to 0 and then load the respective start value again. Each time a timer reaches 0, it generates a trigger pulse and sets the timeout interrupt flag.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number.

23.3.6.5 void PIT_DRV_StopTimer (*uint32_t instance, uint32_t channel*)

This function stops every timer counting. Timers reload their periods respectively after the next time they call the PIT_DRV_StartTimer.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number.

23.3.6.6 void PIT_DRV_SetTimerPeriodByUs (*uint32_t instance, uint32_t channel, uint32_t us*)

The period range depends on the frequency of the PIT source clock. If the required period is out of range, use the lifetime timer if applicable. This function is only valid for one single channel. If channels are chained together, the period here makes no sense.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number.
<i>us</i>	Timer period in microseconds.

23.3.6.7 *uint32_t PIT_DRV_GetTimerPeriodByUs (*uint32_t instance, uint32_t channel*)*

PIT Peripheral driver

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number.

Returns

Timer period in microseconds.

23.3.6.8 `uint32_t PIT_DRV_ReadTimerUs (uint32_t instance, uint32_t channel)`

This function returns an absolute time stamp in microseconds. One common use of this function is to measure the running time of a part of code. Call this function at both the beginning and end of code. The time difference between these two time stamps is the running time. Make sure the running time does not exceed the timer period. The time stamp returned is up-counting.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number.

Returns

Current timer value in microseconds.

23.3.6.9 `void PIT_DRV_SetTimerPeriodByCount (uint32_t instance, uint32_t channel, uint32_t count)`

Timers begin counting from the value set by this function. The counter period of a running timer can be modified by first stopping the timer, setting a new load value, and starting the timer again. If timers are not restarted, the new value is loaded after the next trigger event.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number

<i>count</i>	Timer period in units of count
--------------	--------------------------------

23.3.6.10 `uint32_t PIT_DRV_GetTimerPeriodByCount (uint32_t instance, uint32_t channel)`

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number

Returns

Timer period in units of count

23.3.6.11 `uint32_t PIT_DRV_ReadTimerCount (uint32_t instance, uint32_t channel)`

This function returns the real-time timer counting value, in a range from 0 to a timer period.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number

Returns

Current timer counting value

23.3.6.12 `pit_isr_callback_t PIT_DRV_InstallCallback (uint32_t instance, uint32_t channel, pit_isr_callback_t function)`

System default ISR interfaces are already defined in the `fsl_pit_irq.c`. Users can either edit these ISRs or use this function to register a callback function. The default ISR runs the callback function if there is one installed.

Parameters

<i>instance</i>	PIT module instance number.
<i>channel</i>	Timer channel number.
<i>function</i>	Pointer to pit ISR callback function, pass NULL to uninstall.

PIT Peripheral driver

Returns

Function pointer which was installed before.

23.3.7 Variable Documentation

23.3.7.1 const uint32_t g_pitBaseAddr[]

Chapter 24

Random Number Generator Accelerator (RNGA)

24.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Random Number Generator Accelerator (RNGA) block of Kinetis devices.

Modules

- [RNGA HAL driver](#)
- [RNGA Peripheral Driver](#)

RNGA HAL driver

24.2 RNGA HAL driver

24.2.1 Overview

This section describes the programming interface of the RNGA HAL driver.

Macros

- `#define MAX_COUNT 4096`
the max cpu clock cycles rnga module used to get a new random data

Enumerations

- enum `rnga_mode_t` {
 `kRNGAModeNormal` = 0U,
 `kRNGAModeSleep` = 1U }
RNGA working mode.
- enum `rnga_output_reg_level_t` {
 `kRNGAOutputRegLevelNowords` = 0U,
 `kRNGAOutputRegLevelOneword` = 1U }
Defines the value of output register level.
- enum `rnga_status_t` {
 `kStatus_RNGA_Success` = 0U,
 `kStatus_RNGA_InvalidArgument` = 1U,
 `kStatus_RNGA_Underflow` = 2U,
 `kStatus_RNGA_Timeout` = 3U }
Status structure for RNGA.

RNGA HAL.

- static void `RNGA_HAL_Init` (uint32_t baseAddr)
Initializes the RNGA module.
- static void `RNGA_HAL_Enable` (uint32_t baseAddr)
Enables the RNGA module.
- static void `RNGA_HAL_Disable` (uint32_t baseAddr)
Disables the RNGA module.
- static void `RNGA_HAL_SetHighAssuranceCmd` (uint32_t baseAddr, bool enable)
Sets the RNGA high assurance.
- static void `RNGA_HAL_SetIntMaskCmd` (uint32_t baseAddr, bool enable)
Sets the RNGA interrupt mask.
- static void `RNGA_HAL_ClearIntFlag` (uint32_t baseAddr, bool enable)
Clears the RNGA interrupt.
- static void `RNGA_HAL_SetWorkModeCmd` (uint32_t baseAddr, `rnga_mode_t` mode)
Sets the RNGA in sleep mode or normal mode.
- static uint8_t `RNGA_HAL_GetOutputRegSize` (uint32_t baseAddr)
Gets the output register size.

- static `rnga_output_reg_level_t RNGA_HAL_GetOutputRegLevel` (uint32_t baseAddr)
Gets the output register level.
- static `rnga_mode_t RNGA_HAL_GetWorkMode` (uint32_t baseAddr)
Gets the RNGA working mode.
- static bool `RNGA_HAL_GetErrorIntCmd` (uint32_t baseAddr)
Gets the RNGA status whether an error interrupt has occurred.
- static bool `RNGA_HAL_GetOutputRegUnderflowCmd` (uint32_t baseAddr)
Gets the RNGA status whether an output register underflow has occurred.
- static bool `RNGA_HAL_GetLastReadStatusCmd` (uint32_t baseAddr)
Gets the most recent RNGA read status.
- static bool `RNGA_HAL_GetSecurityViolationCmd` (uint32_t baseAddr)
Gets the RNGA status whether a security violation has occurred.
- static uint32_t `RNGA_HAL_ReadRandomData` (uint32_t baseAddr)
Gets a random data from the RNGA.
- `rnga_status_t RNGA_HAL_GetRandomData` (uint32_t baseAddr, uint32_t *data)
Get random data.
- static void `RNGA_HAL_WriteSeed` (uint32_t baseAddr, uint32_t data)
Inputs an entropy value used to seed the RNGA.

24.2.2 Enumeration Type Documentation

24.2.2.1 enum `rnga_mode_t`

Enumerator

kRNGAModeNormal Normal Mode.

kRNGAModeSleep Sleep Mode.

24.2.2.2 enum `rnga_output_reg_level_t`

Enumerator

kRNGAOutputRegLevelNowords output register no words.

kRNGAOutputRegLevelOneword output register one word.

24.2.2.3 enum `rnga_status_t`

This structure holds the return code of RNGA module.

Enumerator

kStatus_RNGA_Success Success.

kStatus_RNGA_InvalidArgument Invalid argument.

kStatus_RNGA_Underflow Underflow.

kStatus_RNGA_Timeout Timeout.

24.2.3 Function Documentation

24.2.3.1 static void RNGA_HAL_Init(uint32_t *baseAddr*) [inline], [static]

This function initializes the RNGA to a default state.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

24.2.3.2 static void RNGA_HAL_Enable(uint32_t *baseAddr*) [inline], [static]

This function enables the RNGA random data generation and loading.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

24.2.3.3 static void RNGA_HAL_Disable(uint32_t *baseAddr*) [inline], [static]

This function disables the RNGA module.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

24.2.3.4 static void RNGA_HAL_SetHighAssuranceCmd(uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function sets the RNGA high assurance(notification of security violations).

Parameters

<i>baseAddr;RNG-A</i>	base address
<i>enable,0</i>	means notification of security violations disabled. 1 means notification of security violations enabled.

24.2.3.5 static void RNGA_HAL_SetIntMaskCmd(uint32_t *baseAddr*, bool *enable*) [inline], [static]

This function sets the RNGA error interrupt mask.

RNGA HAL driver

Parameters

<i>baseAddr;RNG-A</i>	base address
<i>enable,0</i>	means unmask RNGA interrupt. 1 means mask RNGA interrupt.

24.2.3.6 static void RNGA_HAL_ClearIntFlag (*uint32_t baseAddr, bool enable*) [**inline**], [**static**]

This function clears the RNGA interrupt.

Parameters

<i>baseAddr;RNG-A</i>	base address
<i>enable,0</i>	means do not clear the interrupt. 1 means clear the interrupt.

24.2.3.7 static void RNGA_HAL_SetWorkModeCmd (*uint32_t baseAddr, rnga_mode_t mode*) [**inline**], [**static**]

This function specifies whether the RNGA is in sleep mode or normal mode.

Parameters

<i>baseAddr;RNG-A</i>	base address
<i>mode,kRNGA-ModeNormal</i>	means set RNGA in normal mode. kRNGAModeSleep means set RNGA in sleep mode.

24.2.3.8 static uint8_t RNGA_HAL_GetOutputRegSize (*uint32_t baseAddr*) [**inline**], [**static**]

This function gets the size of the output register as 32-bit random data words it can hold.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

1 means one word(this value is fixed).

24.2.3.9 static rnga_output_reg_level_t RNGA_HAL_GetOutputRegLevel (uint32_t *baseAddr*) [inline], [static]

This function gets the number of random-data words that are in OR [RANDOUT], which indicates if OR is valid.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

0 means no words(empty), 1 means one word(valid).

24.2.3.10 static rnga_mode_t RNGA_HAL_GetWorkMode (uint32_t *baseAddr*) [inline], [static]

This function checks whether the RNGA works in sleep mode or normal mode.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

Kmode_RNGA_Normal means in normal mode Kmode_RNGA_Sleep means in sleep mode

24.2.3.11 static bool RNGA_HAL_GetErrorIntCmd (uint32_t *baseAddr*) [inline], [static]

This function gets the RNGA status whether an OR underflow condition has occurred since the error interrupt was last cleared or the RNGA was reset.

RNGA HAL driver

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

0 means no underflow, 1 means underflow

24.2.3.12 static bool RNGA_HAL_GetOutputRegUnderflowCmd (uint32_t *baseAddr*) [inline], [static]

This function gets the RNGA status whether an OR underflow condition has occurred since the register (SR) was last read or the RNGA was reset.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

0 means no underflow, 1 means underflow

24.2.3.13 static bool RNGA_HAL_GetLastReadStatusCmd (uint32_t *baseAddr*) [inline], [static]

This function gets the RNGA status whether the most recent read of OR[RANDOUT] causes an OR underflow condition.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

0 means no underflow, 1 means underflow

24.2.3.14 static bool RNGA_HAL_GetSecurityViolationCmd (uint32_t *baseAddr*) [inline], [static]

This function gets the RNGA status whether a security violation has occurred when high assurance is enabled.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

0 means no security violation, 1 means security violation

24.2.3.15 static uint32_t RNGA_HAL_ReadRandomData (uint32_t *baseAddr*) [inline], [static]

This function gets a random data from RNGA.

Parameters

<i>baseAddr;RNG-A</i>	base address
-----------------------	--------------

Returns

random data obtained

24.2.3.16 *rnga_status_t* RNGA_HAL_GetRandomData (uint32_t *baseAddr*, uint32_t * *data*)

This function is used to get a random data from RNGA

Parameters

<i>baseAddr;RNG-A</i>	base address
<i>data,pointer</i>	address used to store random data

Returns

one random data

24.2.3.17 static void RNGA_HAL_WriteSeed (uint32_t *baseAddr*, uint32_t *data*) [inline], [static]

This function specifies an entropy value that RNGA uses with its ring oscillations to seed its pseudorandom algorithm.

RNGA HAL driver

Parameters

<i>baseAddr,RNG-A</i>	base address data, external entropy value
-----------------------	---

24.3 RNGA Peripheral Driver

24.3.1 Overview

This section describes the programming interface of the RNGA Peripheral driver. The RNGA driver provides an easy way to initialize and configure the RNGA.

24.3.2 RNGA Initialization

1. To initialize the RNGA module, call the [RNGA_DRV_Init\(\)](#) function and pass in the user configuration structure. This function automatically enables the RNGA module and clock.
2. After calling the [RNGA_DRV_Init\(\)](#) function, the RNGA is enabled and the counter starts working.

This example code shows how to initialize and configure the driver:

```
// Define device configuration.
const rnga_user_config_t init =
{
    .isIntMasked = true, // Mask RNGA Error Interrupt //
    .highAssuranceEnable = true, // Enable high assurance //
};

// Initialize RNGA.
RNGA_DRV_Init(&init);
```

24.3.3 RNGA Set/Get Working Mode

The RNGA works either in sleep mode or normal mode

1. [RNGA_DRV_SetMode\(\)](#) Set RNGA mode.
2. [RNGA_DRV_GetMode\(\)](#) Get RNGA working mode.

24.3.4 Get random data from RNGA

1. [RNGA_DRV_GetRandomData\(\)](#) function gets random data from the RNGA module.

24.3.5 Seed RNGA

1. [RNGA_DRV_Seed\(\)](#) function inputs an entropy value that the RNGA can use to seed the pseudo random algorithm.

24.3.6 RNGA interrupt

If the RNGA interrupt is enabled, the RNGA asserts an interrupt when an underflow error happens.

RNGA Peripheral Driver

1. Enable the RNGA interrupt with the `rnga_user_config_t.isIntMasked` = false.
2. Define the RNGA IRQ function.

```
void RNGA_IRQHandler()
{
    // Enter RNGA ISR //
}
```

Data Structures

- struct `rnga_user_config_t`
Data structure for the RNGA initialization. [More...](#)

Functions

- `rnga_status_t RNGA_DRV_Init` (uint32_t instance, const `rnga_user_config_t` *config)
Initializes the RNGA.
- void `RNGA_DRV_Deinit` (uint32_t instance)
Shuts down the RNGA.
- static void `RNGA_DRV_SetMode` (uint32_t instance, `rnga_mode_t` mode)
Sets the RNGA in normal mode or sleep mode.
- static `rnga_mode_t RNGA_DRV_GetMode` (uint32_t instance)
Gets the RNGA working mode.
- static `rnga_status_t RNGA_DRV_GetRandomData` (uint32_t instance, uint32_t *data)
Gets random data.
- static void `RNGA_DRV_Seed` (uint32_t instance, uint32_t seed)
Feeds the RNGA module.
- void `RNGA_DRV_IRQHandler` (uint32_t instance)
RNGA interrupt handler.

Variables

- const uint32_t `g_rngaBaseAddr` []
Table of base addresses for RNGA instances.
- const IRQn_Type `g_rngaIrqId` [HW_RNG_INSTANCE_COUNT]
Table to save RNGA IRQ enumeration numbers defined in the CMSIS header file.

24.3.7 Data Structure Documentation

24.3.7.1 struct `rnga_user_config_t`

This structure is used when initializing the RNGA by calling the `rnga_init` function. It contains all RNGA configurations.

Data Fields

- bool **isIntMasked**
Mask the triggering of error interrupt.
- bool **highAssuranceEnable**
Enable notification of security violations.

24.3.8 Function Documentation

24.3.8.1 **rnga_status_t RNGA_DRV_Init (uint32_t *instance*, const rnga_user_config_t * *config*)**

This function initializes the RNGA. When called, the RNGA runs immediately according to the specifications in the configuration.

Parameters

<i>instance,RNGA</i>	instance ID
<i>config,pointer</i>	to initialize configuration structure

Returns

RNGA status

24.3.8.2 **void RNGA_DRV_Deinit (uint32_t *instance*)**

This function shuts down the RNGA.

Parameters

<i>instance,RNGA</i>	instance ID
----------------------	-------------

24.3.8.3 **static void RNGA_DRV_SetMode (uint32_t *instance*, rnga_mode_t *mode*) [inline], [static]**

This function sets the RNGA in sleep mode or normal mode.

Parameters

RNGA Peripheral Driver

<i>instance,RNGA</i>	instance ID
<i>mode,normal</i>	mode or sleep mode

24.3.8.4 static **rnga_mode_t RNGA_DRV_GetMode (uint32_t instance) [inline], [static]**

This function gets the RNGA working mode.

Parameters

<i>instance,RNGA</i>	instance ID
----------------------	-------------

Returns

normal mode or sleep mode

24.3.8.5 static **rnga_status_t RNGA_DRV_GetRandomData (uint32_t instance, uint32_t * data) [inline], [static]**

This function gets random data from the RNGA.

Parameters

<i>instance,RNGA</i>	instance ID
<i>data,pointer</i>	address used to store random data

Returns

one random data

24.3.8.6 static void **RNGA_DRV_Seed (uint32_t instance, uint32_t seed) [inline], [static]**

This function inputs an entropy value that the RNGA uses to seed its pseudo-random algorithm.

Parameters

<i>instance,RNGA</i>	instance ID
<i>seed,input</i>	seed value

24.3.8.7 void RNGA_DRV_IRQHandler (uint32_t *instance*)

This function handles the error interrupt caused by the RNGA underflow.

Parameters

<i>instance,RNGA</i>	instance ID
----------------------	-------------

24.3.9 Variable Documentation

24.3.9.1 const uint32_t g_rngabaseAddr[]

24.3.9.2 const IRQn_Type g_rngalrqld[HW_RNG_INSTANCE_COUNT]

Chapter 25

Real Time Clock (RTC)

25.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Real Time Clock (RTC) block of Kinetis devices.

Modules

- [RTC HAL driver](#)
- [RTC Peripheral Driver](#)

25.2 RTC HAL driver

25.2.1 Overview

This section describes the programming interface of the RTC HAL driver. The RTC HAL driver initializes the RTC registers and provides functions to read or modify the RTC registers. These are mostly invoked by the RTC Peripheral driver.

25.2.2 RTC Initialization

The HAL driver provides the enable ([RTC_HAL_Enable\(\)](#)) and disable ([RTC_HAL_Disable\(\)](#)) functions to enable or disable the RTC oscillator or the 32KHz clock from the System oscillator if the platform does not have a RTC oscillator. It also provides an initialization function ([RTC_HAL_Init\(\)](#)) to reset the RTC module.

25.2.3 RTC Setting and reading the RTC time

The HAL driver provides [RTC_HAL_SetDatetime\(\)](#) and [RTC_HAL_GetDatetime\(\)](#) functions to set and read the date and time using an instantiation of the [rtc_datetime_t](#) structure. Details about this structure are provided here.

```
typedef struct RtcDatetime
{
    uint16_t year;
    uint16_t month;
    uint16_t day;
    uint16_t hour;
    uint16_t minute;
    uint8_t second;
} rtc_datetime_t;
```

The HAL driver also provides the ability to set and read the date and time in seconds using the [RTC_HAL_SetDatetimeInsecs\(\)](#) and [RTC_HAL_GetDatetimeInsecs\(\)](#) functions.

NOTE: If the RTC counter clock is not 32KHz then the [RTC_HAL_SetDatetime\(\)](#) and [RTC_HAL_GetDatetime\(\)](#) will not provide accurate results. On such boards, the user should use the [RTC_HAL_SetDatetimeInsecs\(\)](#) and [RTC_HAL_GetDatetimeInsecs\(\)](#) and adjust their seconds calculation taking into account the clock source difference. The RTC peripheral driver adjusts its seconds calculation if the clock feeding RTC counter clock is not 32KHz. The user could use the peripheral driver instead of calling the HAL functions directly.

25.2.4 RTC Setting and reading the Alarm

The HAL driver provides [RTC_HAL_SetAlarm\(\)](#) and [RTC_HAL_GetAlarm\(\)](#) functions to set an alarm and read back the alarm time.

Data Structures

- struct `rtc_datetime_t`
Structure is used to hold the time in a simple "date" format. [More...](#)

RTC HAL API Functions

- void `RTC_HAL_Enable` (uint32_t rtcBaseAddr)
Initializes the RTC module.
- void `RTC_HAL_Disable` (uint32_t rtcBaseAddr)
Disables the RTC module.
- void `RTC_HAL_Init` (uint32_t rtcBaseAddr)
Resets the RTC module.
- void `RTC_HAL_ConvertSecsToDatetime` (const uint32_t *seconds, `rtc_datetime_t` *datetime)
Converts seconds to date time format data structure.
- bool `RTC_HAL_IsDatetimeCorrectFormat` (const `rtc_datetime_t` *datetime)
Checks whether the date time structure elements have the information that is within the range.
- void `RTC_HAL_ConvertDatetimeToSecs` (const `rtc_datetime_t` *datetime, uint32_t *seconds)
Converts the date time format data structure to seconds.
- void `RTC_HAL_SetDatetime` (uint32_t rtcBaseAddr, const `rtc_datetime_t` *datetime)
Sets the RTC date and time according to the given time structure.
- void `RTC_HAL_SetDatetimeInsecs` (uint32_t rtcBaseAddr, const uint32_t seconds)
Sets the RTC date and time according to the given time provided in seconds.
- void `RTC_HAL_GetDatetime` (uint32_t rtcBaseAddr, `rtc_datetime_t` *datetime)
Gets the RTC time and stores it in the given time structure.
- void `RTC_HAL_GetDatetimeInSecs` (uint32_t rtcBaseAddr, uint32_t *seconds)
Gets the RTC time and returns it in seconds.
- void `RTC_HAL_GetAlarm` (uint32_t rtcBaseAddr, `rtc_datetime_t` *date)
Reads the value of the time alarm.
- bool `RTC_HAL_SetAlarm` (uint32_t rtcBaseAddr, const `rtc_datetime_t` *date)
Sets the RTC alarm time and enables the alarm interrupt.

RTC register access functions

- static uint32_t `RTC_HAL_GetSecsReg` (uint32_t rtcBaseAddr)
Reads the value of the time seconds counter.
- static void `RTC_HAL_SetSecsReg` (uint32_t rtcBaseAddr, const uint32_t seconds)
Writes to the time seconds counter.
- static void `RTC_HAL_SetAlarmReg` (uint32_t rtcBaseAddr, const uint32_t seconds)
Sets the time alarm and clears the time alarm flag.
- static uint32_t `RTC_HAL_GetAlarmReg` (uint32_t rtcBaseAddr)
Gets the time alarm register contents.
- static uint16_t `RTC_HAL_GetPrescaler` (uint32_t rtcBaseAddr)
Reads the value of the time prescaler.
- static void `RTC_HAL_SetPrescaler` (uint32_t rtcBaseAddr, const uint16_t prescale)
Sets the time prescaler.
- static uint32_t `RTC_HAL_GetCompensationReg` (uint32_t rtcBaseAddr)
Reads the time compensation register contents.

RTC HAL driver

- static void **RTC_HAL_SetCompensationReg** (uint32_t rtcBaseAddr, const uint32_t compValue)
Writes the value to the RTC TCR register.
- static uint8_t **RTC_HAL_GetCompensationIntervalCounter** (uint32_t rtcBaseAddr)
Reads the current value of the compensation interval counter, which is the field CIC in the RTC TCR register.
- static uint8_t **RTC_HAL_GetTimeCompensationValue** (uint32_t rtcBaseAddr)
Reads the current value used by the compensation logic for the present second interval.
- static uint8_t **RTC_HAL_GetCompensationIntervalRegister** (uint32_t rtcBaseAddr)
Reads the compensation interval register.
- static void **RTC_HAL_SetCompensationIntervalRegister** (uint32_t rtcBaseAddr, const uint8_t value)
Writes the compensation interval.
- static uint8_t **RTC_HAL_GetTimeCompensationRegister** (uint32_t rtcBaseAddr)
Reads the time compensation value which is the configured number of 32.768 kHz clock cycles in each second.
- static void **RTC_HAL_SetTimeCompensationRegister** (uint32_t rtcBaseAddr, const uint8_t compValue)
Writes to the field Time Compensation Register (TCR) of the RTC Time Compensation Register (RTC_TC-R).
- static void **RTC_HAL_SetOsc2pfLoadCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the oscillator configuration for the 2pF load.
- static bool **RTC_HAL_GetOsc2pfLoad** (uint32_t rtcBaseAddr)
Reads the oscillator 2pF load configure bit.
- static void **RTC_HAL_SetOsc4pfLoadCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the oscillator configuration for the 4pF load.
- static bool **RTC_HAL_GetOsc4pfLoad** (uint32_t rtcBaseAddr)
Reads the oscillator 4pF load configure bit.
- static void **RTC_HAL_SetOsc8pfLoadCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the oscillator configuration for the 8pF load.
- static bool **RTC_HAL_GetOsc8pfLoad** (uint32_t rtcBaseAddr)
Reads the oscillator 8pF load configure bit.
- static void **RTC_HAL_SetOsc16pfLoadCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the oscillator configuration for the 16pF load.
- static bool **RTC_HAL_GetOsc16pfLoad** (uint32_t rtcBaseAddr)
Reads the oscillator 16pF load configure bit.
- static void **RTC_HAL_SetClockOutCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the 32 kHz clock output to other peripherals.
- static bool **RTC_HAL_GetClockOutCmd** (uint32_t rtcBaseAddr)
Reads the RTC_CR CLK0 bit.
- static void **RTC_HAL_SetOscillatorCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the oscillator.
- static bool **RTC_HAL_IsOscillatorEnabled** (uint32_t rtcBaseAddr)
Reads the RTC_CR OSCE bit.
- static void **RTC_HAL_SetUpdateModeCmd** (uint32_t rtcBaseAddr, bool lock)
Enables/disables the update mode.
- static bool **RTC_HAL_GetUpdateMode** (uint32_t rtcBaseAddr)
Reads the RTC_CR update mode bit.
- static void **RTC_HAL_SetSupervisorAccessCmd** (uint32_t rtcBaseAddr, bool enableRegWrite)
Enables/disables the supervisor access.
- static bool **RTC_HAL_GetSupervisorAccess** (uint32_t rtcBaseAddr)
Reads the RTC_CR SUP bit.

- static void **RTC_HAL_SoftwareReset** (uint32_t rtcBaseAddr)
Performs a software reset on the RTC module.
- static void **RTC_HAL_SoftwareResetFlagClear** (uint32_t rtcBaseAddr)
Clears the software reset flag.
- static bool **RTC_HAL_ReadSoftwareResetStatus** (uint32_t rtcBaseAddr)
Reads the RTC_CR SWR bit.
- static bool **RTC_HAL_IsCounterEnabled** (uint32_t rtcBaseAddr)
Reads the time counter status (enabled/disabled).
- static void **RTC_HAL_EnableCounter** (uint32_t rtcBaseAddr, bool enable)
Changes the time counter status.
- static bool **RTC_HAL_HasAlarmOccured** (uint32_t rtcBaseAddr)
Checks whether the configured time alarm has occurred.
- static bool **RTC_HAL_HasCounterOverflowed** (uint32_t rtcBaseAddr)
Checks whether a counter overflow has occurred.
- static bool **RTC_HAL_IsTimeInvalid** (uint32_t rtcBaseAddr)
Checks whether the time has been marked as invalid.
- static void **RTC_HAL_SetLockRegistersCmd** (uint32_t rtcBaseAddr, hw_rtc_lr_t bitfields)
Configures the register lock to other module fields.
- static bool **RTC_HAL_GetLockRegLock** (uint32_t rtcBaseAddr)
Obtains the lock status of the lock register.
- static void **RTC_HAL_SetLockRegLock** (uint32_t rtcBaseAddr, bool lock)
Changes the lock status of the lock register.
- static bool **RTC_HAL_GetStatusRegLock** (uint32_t rtcBaseAddr)
Obtains the state of the status register lock.
- static void **RTC_HAL_SetStatusRegLock** (uint32_t rtcBaseAddr, bool lock)
Changes the state of the status register lock.
- static bool **RTC_HAL_GetControlRegLock** (uint32_t rtcBaseAddr)
Obtains the state of the control register lock.
- static void **RTC_HAL_SetControlRegLock** (uint32_t rtcBaseAddr, bool lock)
Changes the state of the control register lock.
- static bool **RTC_HAL_GetTimeCompLock** (uint32_t rtcBaseAddr)
Obtains the state of the time compensation lock.
- static void **RTC_HAL_SetTimeCompLock** (uint32_t rtcBaseAddr, bool lock)
Changes the state of the time compensation lock.
- static bool **RTC_HAL_IsSecsIntEnabled** (uint32_t rtcBaseAddr)
Checks whether the Time Seconds Interrupt is enabled/disabled.
- static void **RTC_HAL_SetSecsIntCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the Time Seconds Interrupt.
- static bool **RTC_HAL_ReadAlarmInt** (uint32_t rtcBaseAddr)
Checks whether the Time Alarm Interrupt is enabled/disabled.
- static void **RTC_HAL_SetAlarmIntCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the Time Alarm Interrupt.
- static bool **RTC_HAL_ReadTimeOverflowInt** (uint32_t rtcBaseAddr)
Checks whether the Time Overflow Interrupt is enabled/disabled.
- static void **RTC_HAL_SetTimeOverflowIntCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the Time Overflow Interrupt.
- static bool **RTC_HAL_ReadTimeInvalidInt** (uint32_t rtcBaseAddr)
Checks whether the Time Invalid Interrupt is enabled/disabled.
- static void **RTC_HAL_SetTimeInvalidIntCmd** (uint32_t rtcBaseAddr, bool enable)
Enables/disables the Time Invalid Interrupt.

25.2.5 Data Structure Documentation

25.2.5.1 struct rtc_datetime_t

Data Fields

- `uint16_t year`
Range from 1970 to 2099.
- `uint16_t month`
Range from 1 to 12.
- `uint16_t day`
Range from 1 to 31 (depending on month).
- `uint16_t hour`
Range from 0 to 23.
- `uint16_t minute`
Range from 0 to 59.
- `uint8_t second`
Range from 0 to 59.

25.2.5.1.0.51 Field Documentation

25.2.5.1.0.51.1 `uint16_t rtc_datetime_t::year`

25.2.5.1.0.51.2 `uint16_t rtc_datetime_t::month`

25.2.5.1.0.51.3 `uint16_t rtc_datetime_t::day`

25.2.5.1.0.51.4 `uint16_t rtc_datetime_t::hour`

25.2.5.1.0.51.5 `uint16_t rtc_datetime_t::minute`

25.2.5.1.0.51.6 `uint8_t rtc_datetime_t::second`

25.2.6 Function Documentation

25.2.6.1 `void RTC_HAL_Enable (uint32_t rtcBaseAddr)`

This function enables the RTC oscillator.

Parameters

<code>rtcBaseAddr</code>	The RTC base address.
--------------------------	-----------------------

25.2.6.2 `void RTC_HAL_Disable (uint32_t rtcBaseAddr)`

This function disables the RTC counter and oscillator.

Parameters

<i>rtcBaseAddr</i>	The RTC base address.
--------------------	-----------------------

25.2.6.3 void RTC_HAL_Init (*uint32_t rtcBaseAddr*)

This function initiates a soft-reset of the RTC module to reset the RTC registers.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

25.2.6.4 void RTC_HAL_ConvertSecsToDatetime (*const uint32_t * seconds*, *rtc_datetime_t * datetime*)

Parameters

<i>seconds</i>	holds the date and time information in seconds
<i>datetime</i>	holds the converted information from seconds in date and time format

25.2.6.5 bool RTC_HAL_IsDatetimeCorrectFormat (*const rtc_datetime_t * datetime*)

Parameters

<i>datetime</i>	holds the date and time information that needs to be converted to seconds
-----------------	---

Returns

returns true if the datetime argument has the right format, false otherwise

25.2.6.6 void RTC_HAL_ConvertDatetimeToSecs (*const rtc_datetime_t * datetime*, *uint32_t * seconds*)

Parameters

RTC HAL driver

<i>datetime</i>	holds the date and time information that needs to be converted to seconds
<i>seconds</i>	holds the converted date and time in seconds

25.2.6.7 void RTC_HAL_SetDatetime (*uint32_t rtcBaseAddr, const rtc_datetime_t * datetime*)

The function converts the data from the time structure to seconds and writes the seconds value to the RTC register. The RTC counter is started after setting the time.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>datetime</i>	[in] Pointer to structure where the date and time details to set are stored.

25.2.6.8 void RTC_HAL_SetDatetimeInsecs (*uint32_t rtcBaseAddr, const uint32_t seconds*)

The RTC counter is started after setting the time.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>seconds</i>	[in] Time in seconds

25.2.6.9 void RTC_HAL_GetDatetime (*uint32_t rtcBaseAddr, rtc_datetime_t * datetime*)

The function reads the value in seconds from the RTC register. It then converts to the time structure which provides the time in date, hour, minutes and seconds.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>datetime</i>	[out] pointer to a structure where the date and time details are stored.

25.2.6.10 void RTC_HAL_GetDatetimeInSecs (*uint32_t rtcBaseAddr, uint32_t * seconds*)

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>seconds</i>	[out] pointer to variable where the RTC time is stored in seconds

25.2.6.11 void RTC_HAL_GetAlarm (**uint32_t rtcBaseAddr, rtc_datetime_t * date**)

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>date</i>	[out] pointer to a variable where the alarm date and time details are stored.

25.2.6.12 bool RTC_HAL_SetAlarm (**uint32_t rtcBaseAddr, const rtc_datetime_t * date**)

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

<i>rtcBaseAddr</i>	The RTC base address..
<i>date</i>	[in] pointer to structure where the alarm date and time details will be stored at.

Returns

true: success in setting the RTC alarm
false: error in setting the RTC alarm.

25.2.6.13 static uint32_t RTC_HAL_GetSecsReg (**uint32_t rtcBaseAddr**) [inline], [static]

The time counter reads as zero if either the SR[TOF] or the SR[TIF] is set.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

contents of the seconds register.

RTC HAL driver

25.2.6.14 static void RTC_HAL_SetSecsReg (*uint32_t rtcBaseAddr*, *const uint32_t seconds*) [inline], [static]

When the time counter is enabled, the TSR is read only and increments once every second provided the SR[TOF] or SR[TIF] is not set. When the time counter is disabled, the TSR can be read or written. Writing to the TSR when the time counter is disabled clears the SR[TOF] and/or the SR[TIF]. Writing to the TSR register with zero is supported, but not recommended, since the TSR reads as zero when either the SR[TIF] or the SR[TOF] is set (indicating the time is invalid).

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
<i>seconds</i>	[in] seconds value.

25.2.6.15 static void RTC_HAL_SetAlarmReg (*uint32_t rtcBaseAddr*, *const uint32_t seconds*) [inline], [static]

When the time counter is enabled, the SR[TAF] is set whenever the TAR[TAR] equals the TSR[TSR] and the TSR[TSR] increments. Writing to the TAR clears the SR[TAF].

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
<i>seconds</i>	[in] alarm value in seconds.

25.2.6.16 static uint32_t RTC_HAL_GetAlarmReg (*uint32_t rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

contents of the alarm register.

25.2.6.17 static uint16_t RTC_HAL_GetPrescaler (*uint32_t rtcBaseAddr*) [inline], [static]

The time counter reads as zero when either the SR[TOF] or the SR[TIF] is set.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

contents of the time prescaler register.

25.2.6.18 static void RTC_HAL_SetPrescaler (*uint32_t rtcBaseAddr*, *const uint16_t prescale*) [inline], [static]

When the time counter is enabled, the TPR is read only and increments every 32.768 kHz clock cycle. When the time counter is disabled, the TPR can be read or written. The TSR[TSR] increments when bit 14 of the TPR transitions from a logic one to a logic zero.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>prescale</i>	Prescaler value

25.2.6.19 static uint32_t RTC_HAL_GetCompensationReg (*uint32_t rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

time compensation register contents.

25.2.6.20 static void RTC_HAL_SetCompensationReg (*uint32_t rtcBaseAddr*, *const uint32_t compValue*) [inline], [static]

Parameters

RTC HAL driver

<i>rtcBaseAddr</i>	The RTC base address
<i>compValue</i>	value to be written to the compensation register.

25.2.6.21 static uint8_t RTC_HAL_GetCompensationIntervalCounter (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

compensation interval value.

25.2.6.22 static uint8_t RTC_HAL_GetTimeCompensationValue (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

time compensation value

25.2.6.23 static uint8_t RTC_HAL_GetCompensationIntervalRegister (uint32_t *rtcBaseAddr*) [inline], [static]

The value is the configured compensation interval in seconds from 1 to 256 to control how frequently the time compensation register should adjust the number of 32.768 kHz cycles in each second. The value is one less than the number of seconds (for example, zero means a configuration for a compensation interval of one second).

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

compensation interval in seconds.

25.2.6.24 static void RTC_HAL_SetCompensationIntervalRegister (uint32_t *rtcBaseAddr*, const uint8_t *value*) [inline], [static]

This configures the compensation interval in seconds from 1 to 256 to control how frequently the TCR should adjust the number of 32.768 kHz cycles in each second. The value written should be one less than the number of seconds (for example, write zero to configure for a compensation interval of one second). This register is double buffered and writes do not take affect until the end of the current compensation interval.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
<i>value</i>	the compensation interval value.

25.2.6.25 static uint8_t RTC_HAL_GetTimeCompensationRegister (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

time compensation value.

25.2.6.26 static void RTC_HAL_SetTimeCompensationRegister (uint32_t *rtcBaseAddr*, const uint8_t *compValue*) [inline], [static]

Configures the number of 32.768 kHz clock cycles in each second. This register is double buffered and writes do not take affect until the end of the current compensation interval. 80h Time prescaler register overflows every 32896 clock cycles.

FFh Time prescaler register overflows every 32769 clock cycles.

00h Time prescaler register overflows every 32768 clock cycles.

01h Time prescaler register overflows every 32767 clock cycles.

... ...

7Fh Time prescaler register overflows every 32641 clock cycles.

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>compValue</i>	value of the time compensation.

25.2.6.27 static void RTC_HAL_SetOsc2pfLoadCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables load false: disables load.

25.2.6.28 static bool RTC_HAL_GetOsc2pfLoad (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: 2pF additional load enabled.
false: 2pF additional load disabled.

25.2.6.29 static void RTC_HAL_SetOsc4pfLoadCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables load. false: disables load

25.2.6.30 static bool RTC_HAL_GetOsc4pfLoad (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: 4pF additional load enabled.
false: 4pF additional load disabled.

25.2.6.31 static void RTC_HAL_SetOsc8pfLoadCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables load. false: disables load.

25.2.6.32 static bool RTC_HAL_GetOsc8pfLoad (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: 8pF additional load enabled.
false: 8pF additional load disabled.

25.2.6.33 static void RTC_HAL_SetOsc16pfLoadCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables load. false: disables load.

RTC HAL driver

25.2.6.34 **static bool RTC_HAL_GetOsc16pfLoad (uint32_t *rtcBaseAddr*) [inline],
[static]**

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: 16pF additional load enabled.
false: 16pF additional load disabled.

25.2.6.35 static void RTC_HAL_SetClockOutCmd (*uint32_t rtcBaseAddr, bool enable*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables clock out. false: disables clock out.

25.2.6.36 static bool RTC_HAL_GetClockOutCmd (*uint32_t rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: 32 kHz clock is not output to other peripherals.
false: 32 kHz clock is output to other peripherals.

25.2.6.37 static void RTC_HAL_SetOscillatorCmd (*uint32_t rtcBaseAddr, bool enable*) [inline], [static]

After enabling, waits for the oscillator startup time before enabling the time counter to allow the 32.768 kHz clock time to stabilize.

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables oscillator. false: disables oscillator.

25.2.6.38 static bool RTC_HAL_IsOscillatorEnabled (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: 32.768 kHz oscillator is enabled false: 32.768 kHz oscillator is disabled.

25.2.6.39 static void RTC_HAL_SetUpdateModeCmd (uint32_t *rtcBaseAddr*, bool *lock*) [inline], [static]

This mode allows the time counter enable bit in the SR to be written even when the status register is locked. When set, the time counter enable, can always be written if the TIF (Time Invalid Flag) or TOF (Time Overflow Flag) are set or if the time counter enable is clear. For devices with the monotonic counter it allows the monotonic enable to be written when it is locked. When set, the monotonic enable can always be written if the TIF (Time Invalid Flag) or TOF (Time Overflow Flag) are set or if the monotonic counter enable is clear. For devices with tamper detect it allows the it to be written when it is locked. When set, the tamper detect can always be written if the TIF (Time Invalid Flag) is clear. Note: Tamper and Monotonic features are not available in all MCUs.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>lock</i>	can be true or false true: registers can be written when locked under limited conditions false: registers cannot be written when locked

25.2.6.40 static bool RTC_HAL_GetUpdateMode (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Registers can be written when locked under limited conditions. false: Registers cannot be written when locked.

25.2.6.41 static void RTC_HAL_SetSupervisorAccessCmd (uint32_t *rtcBaseAddr*, bool *enableRegWrite*) [inline], [static]

This configures non-supervisor mode write access to all RTC registers and non-supervisor mode read access to RTC tamper/monotonic registers. Note: Tamper and Monotonic features are NOT available in all MCUs.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
<i>enableReg- Write</i>	can be true or false true: non-supervisor mode write accesses are supported. false: non-supervisor mode write accesses are not supported and generate a bus error.

25.2.6.42 static bool RTC_HAL_GetSupervisorAccess (uint32_t *rtcBaseAddr*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Non-supervisor mode write accesses are supported
false: Non-supervisor mode write accesses are not supported.

25.2.6.43 static void RTC_HAL_SoftwareReset (uint32_t *rtcBaseAddr*) [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 after VBAT POR and by software explicitly clearing it. Note: access control features (RTC_WAR and RTC_RAR registers) are not available in all MCUs.

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

**25.2.6.44 static void RTC_HAL_SoftwareResetFlagClear (uint32_t *rtcBaseAddr*)
[inline], [static]**

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

**25.2.6.45 static bool RTC_HAL_ReadSoftwareResetStatus (uint32_t *rtcBaseAddr*)
[inline], [static]**

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: SWR is set. false: SWR is cleared.

**25.2.6.46 static bool RTC_HAL_IsCounterEnabled (uint32_t *rtcBaseAddr*) [inline],
[static]**

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: time counter is enabled, time seconds register and time prescaler register are not writeable, but increment.

false: time counter is disabled, time seconds register and time prescaler register are writeable, but do not increment.

**25.2.6.47 static void RTC_HAL_EnableCounter (uint32_t *rtcBaseAddr*, bool *enable*)
[inline], [static]**

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: enables the time counter false: disables the time counter.

25.2.6.48 static bool RTC_HAL_HasAlarmOccured (uint32_t *rtcBaseAddr*) [inline], [static]

Reads time alarm flag (TAF). This flag is set when the time alarm register (TAR) equals the time seconds register (TSR) and the TSR increments. This flag is cleared by writing the TAR register.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

true: time alarm has occurred.

false: no time alarm occurred.

25.2.6.49 static bool RTC_HAL_HasCounterOverflowed (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the value of RTC Status Register (RTC_SR), field Time Overflow Flag (TOF). This flag is set when the time counter is enabled and overflows. The TSR and TPR do not increment and read as zero when this bit is set. This flag is cleared by writing the TSR register when the time counter is disabled.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

true: time overflow occurred and time counter is zero.

false: no time overflow occurred.

25.2.6.50 static bool RTC_HAL_IsTimeInvalid (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the value of RTC Status Register (RTC_SR), field Time Invalid Flag (TIF). This flag is set on VB-AT POR or software reset. The TSR and TPR do not increment and read as zero when this bit is set. This

RTC HAL driver

flag is cleared by writing the TSR register when the time counter is disabled.

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

true: time is INVALID and time counter is zero.
false: time is valid.

25.2.6.51 static void RTC_HAL_SetLockRegistersCmd (uint32_t *rtcBaseAddr*, hw_rtc_lr_t *bitfields*) [inline], [static]

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
<i>bitfields</i>	[in] configuration flags: Valid bitfields: LRL: Lock Register Lock SRL: Status Register Lock CRL: Control Register Lock TCL: Time Compensation Lock For MCUs that have the Tamper Detect only: TIL: Tamper Interrupt Lock TTL: Tamper Trim Lock TDL: Tamper Detect Lock TEL: Tamper Enable Lock TTSL: Tamper Time Seconds Lock For MCUs that have the Monotonic Counter only: MCHL: Monotonic Counter High Lock MCLL: Monotonic Counter Low Lock MEL: Monotonic Enable Lock

25.2.6.52 static bool RTC_HAL_GetLockRegLock (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the value of the field Lock Register Lock (LRL) of the RTC Lock Register (RTC_LR).

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: lock register is not locked and writes complete as normal.
false: lock register is locked and writes are ignored.

25.2.6.53 static void RTC_HAL_SetLockRegLock (uint32_t *rtcBaseAddr*, bool *lock*) [inline], [static]

Writes to the field Lock Register Lock (LRL) of the RTC Lock Register (RTC_LR). Once cleared, this can only be set by VBAT POR or software reset.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>lock</i>	can be true or false true: Lock register is not locked and writes complete as normal. false: Lock register is locked and writes are ignored.

25.2.6.54 static bool RTC_HAL_GetStatusRegLock (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the value of field Status Register Lock (SRL) of the RTC Lock Register (RTC_LR), which is the field Status Register.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Status register is not locked and writes complete as normal.
false: Status register is locked and writes are ignored.

25.2.6.55 static void RTC_HAL_SetStatusRegLock (uint32_t *rtcBaseAddr*, bool *lock*) [inline], [static]

Writes to the field Status Register Lock (SRL) of the RTC Lock Register (RTC_LR). Once cleared, this can only be set by VBAT POR or software reset.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>lock</i>	can be true or false true: Status register is not locked and writes complete as normal. false: Status register is locked and writes are ignored.

25.2.6.56 static bool RTC_HAL_GetControlRegLock (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the field Control Register Lock (CRL) value of the RTC Lock Register (RTC_LR).

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Control register is not locked and writes complete as normal.
false: Control register is locked and writes are ignored.

25.2.6.57 static void RTC_HAL_SetControlRegLock (uint32_t *rtcBaseAddr*, bool *lock*) [inline], [static]

Writes to the field Control Register Lock (CRL) of the RTC Lock Register (RTC_LR). Once cleared, this can only be set by VBAT POR or software reset.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>lock</i>	can be true or false true: Control register is not locked and writes complete as normal. false: Control register is locked and writes are ignored.

25.2.6.58 static bool RTC_HAL_GetTimeCompLock (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the field Time Compensation Lock (TCL) value of the RTC Lock Register (RTC_LR).

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Time compensation register is not locked and writes complete as normal.
false: Time compensation register is locked and writes are ignored.

25.2.6.59 static void RTC_HAL_SetTimeCompLock (uint32_t *rtcBaseAddr*, bool *lock*) [inline], [static]

Writes to the field Time Compensation Lock (TCL) of the RTC Lock Register (RTC_LR). Once cleared, this can only be set by VBAT POR or software reset.

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>lock</i>	can be true or false true: Time compensation register is not locked and writes complete as normal. false: Time compensation register is locked and writes are ignored.

25.2.6.60 static bool RTC_HAL_IsSecsIntEnabled (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the value of field Time Seconds Interrupt Enable (TSIE) of the RTC Interrupt Enable Register (RTC_IER). The seconds interrupt is an edge-sensitive interrupt with a dedicated interrupt vector. It is generated once a second and requires no software overhead (there is no corresponding status flag to clear).

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Seconds interrupt is enabled.
false: Seconds interrupt is disabled.

25.2.6.61 static void RTC_HAL_SetSecsIntCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Writes to the field Time Seconds Interrupt Enable (TSIE) of the RTC Interrupt Enable Register (RTC_IER). Note: The seconds interrupt is an edge-sensitive interrupt with a dedicated interrupt vector. It is

generated once a second and requires no software overhead (there is no corresponding status flag to clear).

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: Seconds interrupt is enabled. false: Seconds interrupt is disabled.

25.2.6.62 static bool RTC_HAL_ReadAlarmInt (*uint32_t rtcBaseAddr*) [inline], [static]

Reads the field Time Alarm Interrupt Enable (TAIE) value of the RTC Interrupt Enable Register (RTC_IER).

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

true: Time alarm flag does generate an interrupt.
false: Time alarm flag does not generate an interrupt.

25.2.6.63 static void RTC_HAL_SetAlarmIntCmd (*uint32_t rtcBaseAddr, bool enable*) [inline], [static]

Writes to the field Time Alarm Interrupt Enable (TAIE) of the RTC Interrupt Enable Register (RTC_IER).

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: Time alarm flag does generate an interrupt. false: Time alarm flag does not generate an interrupt.

25.2.6.64 static bool RTC_HAL_ReadTimeOverflowInt (*uint32_t rtcBaseAddr*) [inline], [static]

Reads the field Time Overflow Interrupt Enable (TOIE) of the value of the RTC Interrupt Enable Register (RTC_IER).

Parameters

<i>rtcBaseAddr</i>	The RTC base address..
--------------------	------------------------

Returns

- true: Time overflow flag does generate an interrupt.
- false: Time overflow flag does not generate an interrupt.

25.2.6.65 static void RTC_HAL_SetTimeOverflowIntCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Writes to the field Time Overflow Interrupt Enable (TOIE) of the RTC Interrupt Enable Register (RTC_IER).

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: Time overflow flag does generate an interrupt. false: Time overflow flag does not generate an interrupt.

25.2.6.66 static bool RTC_HAL_ReadTimeInvalidInt (uint32_t *rtcBaseAddr*) [inline], [static]

Reads the value of the field Time Invalid Interrupt Enable (TIIE) of the RTC Interrupt Enable Register (RTC_IER).

Parameters

<i>rtcBaseAddr</i>	The RTC base address
--------------------	----------------------

Returns

- true: Time invalid flag does generate an interrupt.
- false: Time invalid flag does not generate an interrupt.

25.2.6.67 static void RTC_HAL_SetTimeInvalidIntCmd (uint32_t *rtcBaseAddr*, bool *enable*) [inline], [static]

Writes to the field Time Invalid Interrupt Enable (TIIE) of the RTC Interrupt Enable Register (RTC_IER).

RTC HAL driver

Parameters

<i>rtcBaseAddr</i>	The RTC base address
<i>enable</i>	can be true or false true: Time invalid flag does generate an interrupt. false: Time invalid flag does not generate an interrupt.

25.3 RTC Peripheral Driver

25.3.1 Overview

This section describes the programming interface of the RTC Peripheral Driver. The RTC Peripheral driver sets and gets the RTC clock, sets and reads the RTC alarm time, and receives notifications when an alarm is triggered.

25.3.2 RTC Peripheral Driver Initialization

To initialize, the user calls the [RTC_DRV_Init\(\)](#) function with the RTC instance number. Most SoCs have only one RTC instance. Therefore, the instance number is zero. The driver initialization function un-gates the RTC module clock, initializes the RTC HAL layer driver, and enables the RTC interrupts.

This is an example how to create the `rtc_init()` function.

```
// init the rtc module //
RTC_DRV_Init(0);
```

25.3.3 RTCS Setting and reading the RTC time

The RTC driver uses an instantiation of the `rtc_datetime_t` structure either to configure or read the date and time. Call the [RTC_DRV_SetDatetime\(\)](#) function to configure the current date and time and call the [RTC_DRV_GetDatetime\(\)](#) function to read the current date and time at a later time. This example shows how to use these functions.

```
rtc_datetime_t datetimeToSet;

RTC_DRV_Init(0);

datetimeToSet.year = 2013;
datetimeToSet.month = 10;
datetimeToSet.day = 13;
datetimeToSet.hour = 18;
datetimeToSet.minute = 55;
datetimeToSet.second = 30;

// set the datetime //
result = RTC_DRV_SetDatetime(0, &datetime);

// get datetime //
RTC_DRV_GetDatetime(0, &datetime);
printf("Current datetime: %04hd-%02hd-%02hd %02hd:%02hd:%02hd\r\n",
      datetime.year, datetime.month, datetime.day,
      datetime.hour, datetime.minute, datetime.second);
```

25.3.4 RTC Triggering an Alarm

Call the [RTC_DRV_SetAlarm\(\)](#) function to set the alarm time and call the [RTC_DRV_GetAlarm\(\)](#) function to read the configured alarm time. Set the current time using the steps mentioned earlier prior to using

RTC Peripheral Driver

the call to set the alarm time. The user can enable the option to trigger an interrupt when an alarm occurs. This is done by either calling the [RTC_DRV_SetAlarmIntCmd\(\)](#) function or through an argument to the [RTC_DRV_SetAlarm\(\)](#) function. This is an example to set an RTC alarm. This example causes an alarm interrupt to be triggered after 5 minutes.

```
rtc_datetime_t datetimeToSet;
rtc_datetime_t alarmTimeToSet;

RTC_DRV_Init(0);

datetimeToSet.year = 2013;
datetimeToSet.month = 10;
datetimeToSet.day = 13;
datetimeToSet.hour = 18;
datetimeToSet.minute = 55;
datetimeToSet.second = 30;

RTC_DRV_SetDatetime(0, &datetimeToSet);

alarmTimeToSet.year = datetimeToSet.year;
alarmTimeToSet.month = datetimeToSet.month;
alarmTimeToSet.day = datetimeToSet.day;
alarmTimeToSet.minute = datetimeToSet.minute + 5;
alarmTimeToSet.second = datetimeToSet.second;
RTC_DRV_SetAlarm(0, &alarmTimeToSet, true);
```

25.3.5 RTC Repeated alarm

To request a repeated alarm, a configuration structure is available to configure the repeat alarm information. These are the structure details:

```
typedef struct RtcRepeatAlarmState
{
    rtc_datetime_t alarmTime;
    rtc_datetime_t alarmRepTime;
} rtc_repeat_alarm_state_t;
```

Call the [RTC_DRV_InitRepeatAlarm\(\)](#) function and provide the repeat alarm configuration structure to the RTC driver. The user should not free this structure because the RTC driver stores the pointer to this structure to configure the future alarm times.

This is an example to set a RTC repeat alarm. The alarm interrupt is triggered after 5 minutes and every minute after that.

```
rtc_repeat_alarm_state_t alarm_state;

rtc_datetime_t alarmTimeToSet;
rtc_datetime_t alarmReptime;
rtc_datetime_t datetimeToSet;

RTC_DRV_Init(0);
RTC_DRV_InitRepeatAlarm(0, &alarm_state);
RTC_DRV_SetDatetime(0, &datetimeToSet);

alarmTimeToSet.minute = datetimeToSet.minute + 5;
```

```

alarmReptime.year = 0;
alarmReptime.month = 0;
alarmReptime.day = 0;
alarmReptime.hour = 0;
alarmReptime.minute = 1;

if(!RTC_DRV_SetAlarmRepeat(0, &alarmTimeToSet, &alarmReptime))
{
    exit_error();
}

```

25.3.6 RTC Enable and Disable Alarm Interrupts

Call the [RTC_DRV_SetAlarmIntCmd\(\)](#) function to enable or disable the RTC alarm interrupt. Call the [RTC_DRV_GetAlarmIntCmd\(\)](#) function to get the state of the RTC alarm interrupt bit.

25.3.7 RTC Interrupt handler

The RTC driver provides an interrupt handler for the seconds and alarm interrupts. These handlers clear the status bits. When the repeated alarm is requested, the alarm interrupt handler uses the information provided in the `rtc_repeat_alarm_state_t` structure to schedule the next alarm.

To add more actions to the default handler, add calls to the functions inside the interrupt handlers [RTC_I-RQHandler\(\)](#) and [RTC_Seconds_IRQHandler\(\)](#) functions.

Data Structures

- struct [rtc_repeat_alarm_state_t](#)
RTC repeated alarm information used by the RTC driver. [More...](#)

Functions

- bool [RTC_DRV_IsCounterEnabled](#) (uint32_t instance)
Checks whether the RTC is enabled.
- void [RTC_DRV_SetSecsIntCmd](#) (uint32_t instance, bool secondsEnable)
Enables or disables the RTC seconds interrupt.
- void [RTC_DRV_SetAlarmIntCmd](#) (uint32_t instance, bool alarmEnable)
Enables or disables the alarm interrupt.
- bool [RTC_DRV_GetAlarmIntCmd](#) (uint32_t instance)
Reads the alarm interrupt.
- bool [RTC_DRV_IsAlarmPending](#) (uint32_t instance)
Reads the alarm status to see if the alarm has triggered.
- void [RTC_DRV_SetTimeCompensation](#) (uint32_t instance, uint32_t compensationInterval, uint32_t compensationTime)
Writes the compensation value to the RTC compensation register.
- void [RTC_DRV_GetTimeCompensation](#) (uint32_t instance, uint32_t *compensationInterval, uint32_t *compensationTime)

RTC Peripheral Driver

- Reads the compensation value from the RTC compensation register.
- void **RTC_DRV_AlarmIntAction** (uint32_t instance)
Action to take when an RTC alarm interrupt is triggered.
- void **RTC_DRV_SecsIntAction** (uint32_t instance)
Action to take when an RTC seconds interrupt is triggered.

Variables

- const uint32_t **g_RTCBaseAddr** []
Table of base addresses for RTC instances.
- const IRQn_Type **g_RTCIrqId** []
Table to save RTC Alarm IRQ numbers for RTC instances.
- const IRQn_Type **g_RTCSecondsIrqId** []
Table to save RTC Seconds IRQ numbers for RTC instances.

Initialize and Deinitialize

- void **RTC_DRV_Init** (uint32_t instance)
Initializes the RTC module.
- void **RTC_DRV_Deinit** (uint32_t instance)
Disables the RTC module clock gate control.

RTC datetime set and get

- bool **RTC_DRV_SetDatetime** (uint32_t instance, **rtc_datetime_t** *datetime)
Sets the RTC date and time according to the given time structure.
- void **RTC_DRV_GetDatetime** (uint32_t instance, **rtc_datetime_t** *datetime)
Gets the RTC time and stores it in the given time structure.

RTC alarm

- bool **RTC_DRV_SetAlarm** (uint32_t instance, **rtc_datetime_t** *alarmTime, bool enableAlarmInterrupt)
Sets the RTC alarm time and enables the alarm interrupt.
- void **RTC_DRV_GetAlarm** (uint32_t instance, **rtc_datetime_t** *date)
Returns the RTC alarm time.
- void **RTC_DRV_InitRepeatAlarm** (uint32_t instance, **rtc_repeat_alarm_state_t** *repeatAlarmState)
Initializes the RTC repeat alarm state structure.
- bool **RTC_DRV_SetAlarmRepeat** (uint32_t instance, **rtc_datetime_t** *alarmTime, **rtc_datetime_t** *alarmRepInterval)
Sets an alarm that is periodically repeated.
- void **RTC_DRV_DeinitRepeatAlarm** (uint32_t instance)
De-initializes the RTC repeat alarm state structure.

ISR Functions

- void **RTC_IRQHandler** (void)
Implementation of RTC Alarm handler named in startup code.
- void **RTC_Seconds_IRQHandler** (void)
Implementation of RTC Seconds handler named in startup code.

25.3.8 Data Structure Documentation

25.3.8.1 **struct rtc_repeat_alarm_state_t**

Data Fields

- **rtc_datetime_t alarmTime**
Set the RTC alarm time.
- **rtc_datetime_t alarmRepTime**
Period for alarm to repeat, needs alarm interrupt be enabled.

25.3.8.1.0.52 Field Documentation

25.3.8.1.0.52.1 **rtc_datetime_t rtc_repeat_alarm_state_t::alarmTime**

25.3.8.1.0.52.2 **rtc_datetime_t rtc_repeat_alarm_state_t::alarmRepTime**

25.3.9 Function Documentation

25.3.9.1 **void RTC_DRV_Init (uint32_t instance)**

Enables the RTC clock and enables interrupts if requested by the user.

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

25.3.9.2 **void RTC_DRV_Deinit (uint32_t instance)**

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

25.3.9.3 **bool RTC_DRV_IsCounterEnabled (uint32_t instance)**

The function checks the TCE bit in the RTC control register.

RTC Peripheral Driver

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

Returns

- true: The RTC counter is enabled
- false: The RTC counter is disabled

25.3.9.4 **bool RTC_DRV_SetDatetime (uint32_t *instance*, rtc_datetime_t * *datetime*)**

The RTC counter is started after the time is set.

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>datetime</i>	[in] pointer to structure where the date and time details to set are stored.

Returns

- true: success in setting the time and starting the RTC
- false: failure. An error occurs because the datetime format is incorrect.

25.3.9.5 **void RTC_DRV_GetDatetime (uint32_t *instance*, rtc_datetime_t * *datetime*)**

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>datetime</i>	[out] pointer to structure where the date and time details are stored.

25.3.9.6 **void RTC_DRV_SetSecsIntCmd (uint32_t *instance*, bool *secondsEnable*)**

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

<i>secondsEnable</i>	Takes true or false true: indicates seconds interrupt should be enabled false: indicates seconds interrupt should be disabled
----------------------	---

25.3.9.7 **bool RTC_DRV_SetAlarm (uint32_t *instance*, rtc_datetime_t * *alarmTime*, bool *enableAlarmInterrupt*)**

The function checks if the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>alarmTime</i>	[in] pointer to structure where the alarm time is store.
<i>enableAlarm- Interrupt</i>	Takes true or false true: indicates alarm interrupt should be enabled false: indicates alarm interrupt should be disabled

Returns

true: success in setting the RTC alarm
false: error in setting the RTC alarm. Error is because the alarm datetime format is incorrect.

25.3.9.8 **void RTC_DRV_GetAlarm (uint32_t *instance*, rtc_datetime_t * *date*)**

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>date</i>	[out] pointer to structure where the alarm date and time details are stored.

25.3.9.9 **void RTC_DRV_InitRepeatAlarm (uint32_t *instance*, rtc_repeat_alarm_state_t * *repeatAlarmState*)**

The RTC driver uses this user-provided structure to store the alarm state information.

RTC Peripheral Driver

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>repeatAlarm-State</i>	Pointer to structure where the alarm state is stored

25.3.9.10 bool RTC_DRV_SetAlarmRepeat (uint32_t *instance*, rtc_datetime_t * *alarmTime*, rtc_datetime_t * *alarmRepInterval*)

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>alarmTime</i>	Pointer to structure where the alarm time is provided.
<i>alarmRep-Interval</i>	pointer to structure with the alarm repeat interval.

Returns

true: success in setting the RTC alarm

false: error in setting the RTC alarm. Error is because the alarm datetime format is incorrect.

25.3.9.11 void RTC_DRV_DeinitRepeatAlarm (uint32_t *instance*)

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

25.3.9.12 void RTC_DRV_SetAlarmIntCmd (uint32_t *instance*, bool *alarmEnable*)

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>alarmEnable</i>	Takes true or false true: indicates alarm interrupt should be enabled false: indicates alarm interrupt should be disabled

25.3.9.13 bool RTC_DRV_GetAlarmIntCmd (uint32_t *instance*)

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

Returns

true: indicates alarm interrupt is enabled
 false: indicates alarm interrupt is disabled

25.3.9.14 **bool RTC_DRV_IsAlarmPending (uint32_t *instance*)**

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

Returns

returns alarm status, for example, returns whether the alarm triggered
 true: indicates alarm has occurred
 false: indicates alarm has not occurred

25.3.9.15 **void RTC_DRV_SetTimeCompensation (uint32_t *instance*, uint32_t *compensationInterval*, uint32_t *compensationTime*)**

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>compensation-Interval</i>	User specified compensation interval that is written to the CIR field in RTC Time Compensation Register (TCR)
<i>compensation-Time</i>	User specified compensation time that is written to the TCR field in RTC Time Compensation Register (TCR)

25.3.9.16 **void RTC_DRV_GetTimeCompensation (uint32_t *instance*, uint32_t * *compensationInterval*, uint32_t * *compensationTime*)**

RTC Peripheral Driver

Parameters

<i>instance</i>	The RTC peripheral instance number.
<i>compensation-Interval</i>	User specified pointer to store the compensation interval counter. This value is read from the CIC field in RTC Time Compensation Register (TCR)
<i>compensation-Time</i>	User specified pointer to store the compensation time value. This value is read from the TCV field in RTC Time Compensation Register (TCR)

25.3.9.17 void RTC_IRQHandler (void)

Handles the RTC alarm interrupt and invokes any callback that is interested in the RTC alarm

25.3.9.18 void RTC_Seconds_IRQHandler (void)

Handles the RTC seconds interrupt and invokes any callback that is interested in the RTC second tick.

25.3.9.19 void RTC_DRV_AlarmIntAction (uint32_t *instance*)

To receive alarms periodically, the RTC_TAR register is updated using the repeat interval.

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

25.3.9.20 void RTC_DRV_SecsIntAction (uint32_t *instance*)

Disables the time seconds interrupt (TSIE) bit.

Parameters

<i>instance</i>	The RTC peripheral instance number.
-----------------	-------------------------------------

25.3.10 Variable Documentation

25.3.10.1 const uint32_t g_RTCBaseAddr[]

25.3.10.2 const IRQn_Type g_RTCIRQId[]

25.3.10.3 const IRQn_Type g_RTCSecondsIRQId[]

Chapter 26

Synchronous Audio Interface (SAI)

26.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Synchronous Audio Interface (SAI) block of Kinetis devices.

Modules

- [SAI HAL driver](#)
- [SAI Peripheral driver](#)

26.2 SAI HAL driver

26.2.1 Overview

This section describes the programming interface of the SAI HAL driver.

Files

- file [fsl_sai_hal.h](#)

Enumerations

- enum [sai_protocol_t](#)
Define the bus type of sai.
- enum [sai_master_slave_t](#) {
 kSaiMaster = 0x0,
 kSaiSlave = 0x1 }
Master or slave mode.
- enum [sai_clk_polarity_t](#) {
 kSaiClkPolarityHigh = 0x0,
 kSaiClkPolarityLow = 0x1 }
Polarity of SAI clock.
- enum [sai_clk_direction_t](#) {
 kSaiClkInternal = 0x0,
 kSaiClkExternal = 0x1 }
Clock generate direction.
- enum [sai_data_order_t](#) {
 kSaiLSBFirst = 0x0,
 kSaiMSBFirst = 0x1 }
Data transfer polarity, means MSB first of LSB first.
- enum [sai_sync_mode_t](#) {
 kSaiModeAsync = 0x0,
 kSaiModeSync = 0x1,
 kSaiModeSyncWithOtherTx = 0x2,
 kSaiModeSyncWithOtherRx = 0x3 }
Synchronous or asynchronous mode.
- enum [sai_mclk_source_t](#) {
 kSaiMclkSourceSysclk = 0x0,
 kSaiMclkSourceSelect1 = 0x1,
 kSaiMclkSourceSelect2 = 0x2,
 kSaiMclkSourceSelect3 = 0x3 }
Mater clock source.
- enum [sai_bclk_source_t](#) {
 kSaiBclkSourceBusclk = 0x0,
 kSaiBclkSourceMclkDiv = 0x1,
 kSaiBclkSourceOtherSai0 = 0x2,

- ```

kSaiBclkSourceOtherSai1 = 0x3 }

Bit clock source.
• enum sai_interrupt_request_t {
 kSaiIntrequestWordStart = 0x0,
 kSaiIntrequestSyncError = 0x1,
 kSaiIntrequestFIFOWarning = 0x2,
 kSaiIntrequestFIFOError = 0x3,
 kSaiIntrequestFIFOReset = 0x4 }

The SAI state flag.
• enum sai_dma_request_t {
 kSaiDmaReqFIFOWarning = 0x0,
 kSaiDmaReqFIFOReset = 0x1 }

The DMA request sources.
• enum sai_state_flag_t {
 kSaiStateFlagWordStart = 0x0,
 kSaiStateFlagSyncError = 0x1,
 kSaiStateFlagFIFOError = 0x2 ,
 kSaiStateFlagSoftReset = 0x5 }

The SAI state flag.
• enum sai_reset_type_t {
 kSaiResetTypeSoftware = 0x0,
 kSaiResetTypeFIFO = 0x1 }

The reset type.
• enum sai_run_mode_t {
 kSaiRunModeDebug = 0x0,
 kSaiRunModeStop = 0x1 }

```

## Module control

- void **SAI\_HAL\_TxInit** (uint32\_t saiBaseAddr)
 

*Initializes the SAI Tx.*
- void **SAI\_HAL\_RxInit** (uint32\_t saiBaseAddr)
 

*Initializes the SAI Rx.*
- void **SAI\_HAL\_TxSetProtocol** (uint32\_t saiBaseAddr, **sai\_protocol\_t** protocol)
 

*Sets Tx protocol relevant settings.*
- void **SAI\_HAL\_RxSetProtocol** (uint32\_t saiBaseAddr, **sai\_protocol\_t** protocol)
 

*Sets Rx protocol relevant settings.*
- void **SAI\_HAL\_TxSetMasterSlave** (uint32\_t saiBaseAddr, **sai\_master\_slave\_t** master\_slave\_mode)
 

*Sets master or slave mode.*
- void **SAI\_HAL\_RxSetMasterSlave** (uint32\_t saiBaseAddr, **sai\_master\_slave\_t** master\_slave\_mode)
 

*Sets master or slave mode.*

## Master clock configuration

- static void **SAI\_HAL\_SetMclkSrc** (uint32\_t saiBaseAddr, **sai\_mclk\_source\_t** source)
 

*Sets the master clock source.*

## SAI HAL driver

- static uint32\_t **SAI\_HAL\_GetMclkSrc** (uint32\_t saiBaseAddr)  
*Gets the master clock source.*
- static void **SAI\_HAL\_SetMclkDividerCmd** (uint32\_t saiBaseAddr, bool enable)  
*Sets the direction of the SAI master clock.*
- static bool **SAI\_HAL\_GetMclkDivUpdatingCmd** (uint32\_t saiBaseAddr)  
*Flag to see if the master clock divider is re-divided.*

## Bit clock configuration

- static void **SAI\_HAL\_TxSetBclkSrc** (uint32\_t saiBaseAddr, **sai\_bclk\_source\_t** source)  
*Sets the bit clock source of Tx.*
- static void **SAI\_HAL\_RxSetBclkSrc** (uint32\_t saiBaseAddr, **sai\_bclk\_source\_t** source)  
*Sets bit clock source of the Rx.*
- static uint32\_t **SAI\_HAL\_TxGetBclkSrc** (uint32\_t saiBaseAddr)  
*Gets the bit clock source of Tx.*
- static uint32\_t **SAI\_HAL\_RxGetBclkSrc** (uint32\_t saiBaseAddr)  
*Gets bit clock source of the Rx.*
- static void **SAI\_HAL\_TxSetBclkDiv** (uint32\_t saiBaseAddr, uint32\_t divider)  
*Sets the Tx bit clock divider value.*
- static void **SAI\_HAL\_RxSetBclkDiv** (uint32\_t saiBaseAddr, uint32\_t divider)  
*Sets the Rx bit clock divider value.*
- static void **SAI\_HAL\_TxSetBclkCmd** (uint32\_t saiBaseAddr, bool enable)  
*Enables or disables the Tx bit clock.*
- static void **SAI\_HAL\_RxSetBclkCmd** (uint32\_t saiBaseAddr, bool enable)  
*Enables or disables the Rx bit clock.*
- static void **SAI\_HAL\_TxSetBclkInputCmd** (uint32\_t saiBaseAddr, bool enable)  
*Enables or disables the Tx bit clock input bit.*
- static void **SAI\_HAL\_RxSetBclkInputCmd** (uint32\_t saiBaseAddr, bool enable)  
*Enables or disables the Rx bit clock input bit.*
- static void **SAI\_HAL\_TxSetSwapBclkCmd** (uint32\_t saiBaseAddr, bool enable)  
*Sets the Tx bit clock swap.*
- static void **SAI\_HAL\_RxSetSwapBclkCmd** (uint32\_t saiBaseAddr, bool enable)  
*Sets the Rx bit clock swap.*
- static void **SAI\_HAL\_TxSetBclkDir** (uint32\_t saiBaseAddr, **sai\_clk\_direction\_t** direction)  
*Sets the direction of the Tx SAI bit clock.*
- static void **SAI\_HAL\_RxSetBclkDir** (uint32\_t saiBaseAddr, **sai\_clk\_direction\_t** direction)  
*Sets the direction of the Rx SAI bit clock.*
- static void **SAI\_HAL\_TxSetBclkPolarity** (uint32\_t saiBaseAddr, **sai\_clk\_polarity\_t** pol)  
*Sets the polarity of the Tx SAI bit clock.*
- static void **SAI\_HAL\_RxSetBclkPolarity** (uint32\_t saiBaseAddr, **sai\_clk\_polarity\_t** pol)  
*Sets the polarity of the Rx SAI bit clock.*

## Frame sync configuration

- static void **SAI\_HAL\_TxSetFrameSize** (uint32\_t saiBaseAddr, uint32\_t size)  
*Sets the Tx frame size.*
- static void **SAI\_HAL\_RxSetFrameSize** (uint32\_t saiBaseAddr, uint32\_t size)  
*Sets the Rx frame size.*
- static uint32\_t **SAI\_HAL\_TxGetFrameSize** (uint32\_t saiBaseAddr)

- static uint32\_t **SAI\_HAL\_RxGetFrameSize** (uint32\_t saiBaseAddr)
 

*Gets the Tx frame size.*
- static void **SAI\_HAL\_TxSetFrameSyncWidth** (uint32\_t saiBaseAddr, uint32\_t width)
 

*Gets the Tx frame size.*

*Sets the Tx sync width.*
- static void **SAI\_HAL\_RxSetFrameSyncWidth** (uint32\_t saiBaseAddr, uint32\_t width)
 

*Sets the Rx sync width.*
- static void **SAI\_HAL\_TxSetFrameSyncPolarity** (uint32\_t saiBaseAddr, **sai\_clk\_polarity\_t** pol)
 

*Sets the polarity of the Tx frame sync.*
- static void **SAI\_HAL\_RxSetFrameSyncPolarity** (uint32\_t saiBaseAddr, **sai\_clk\_polarity\_t** pol)
 

*Sets the polarity of the Rx frame sync.*
- static void **SAI\_HAL\_TxSetFrameSyncDir** (uint32\_t saiBaseAddr, **sai\_clk\_direction\_t** direction)
 

*Sets the direction of the SAI Tx frame sync.*
- static void **SAI\_HAL\_RxSetFrameSyncDir** (uint32\_t saiBaseAddr, **sai\_clk\_direction\_t** direction)
 

*Sets the direction of the SAI Rx frame sync.*
- static void **SAI\_HAL\_TxSetBitOrder** (uint32\_t saiBaseAddr, **sai\_data\_order\_t** order)
 

*Sets the Tx data transfer order.*
- static void **SAI\_HAL\_RxSetBitOrder** (uint32\_t saiBaseAddr, **sai\_data\_order\_t** order)
 

*Sets the Rx data transfer order.*
- static void **SAI\_HAL\_TxSetFrameSyncEarlyCmd** (uint32\_t saiBaseAddr, bool enable)
 

*Tx Frame sync one bit early.*
- static void **SAI\_HAL\_RxSetFrameSyncEarlyCmd** (uint32\_t saiBaseAddr, bool enable)
 

*Rx Frame sync one bit early.*

## Word configurations

- static void **SAI\_HAL\_TxSetWordSize** (uint32\_t saiBaseAddr, uint32\_t bits)
 

*Sets the word size for Tx.*
- static void **SAI\_HAL\_RxSetWordSize** (uint32\_t saiBaseAddr, uint32\_t bits)
 

*Sets the word size for Rx.*
- static uint32\_t **SAI\_HAL\_TxGetWordSize** (uint32\_t saiBaseAddr)
 

*Gets the Tx word size.*
- static uint32\_t **SAI\_HAL\_RxGetWordSize** (uint32\_t saiBaseAddr)
 

*Gets the Rx word size.*
- static void **SAI\_HAL\_TxSetFirstWordSize** (uint32\_t saiBaseAddr, uint8\_t size)
 

*Sets the size of the first word of the Tx frame .*
- static void **SAI\_HAL\_RxSetFirstWordSize** (uint32\_t saiBaseAddr, uint8\_t size)
 

*Sets the size of the first word of Rx frame .*
- static void **SAI\_HAL\_TxSetFirstBitShifted** (uint32\_t saiBaseAddr, uint32\_t index)
 

*Sets the FIFO index for the first bit data.*
- static void **SAI\_HAL\_RxSetFirstBitShifted** (uint32\_t saiBaseAddr, uint32\_t index)
 

*Sets the index in FIFO for the first bit data.*

### 26.2.2 Enumeration Type Documentation

#### 26.2.2.1 enum sai\_master\_slave\_t

Enumerator

*kSaiMaster* Master mode.

*kSaiSlave* Slave mode.

#### 26.2.2.2 enum sai\_clk\_polarity\_t

Enumerator

*kSaiClkPolarityHigh* Clock active high.

*kSaiClkPolarityLow* Clock active low.

#### 26.2.2.3 enum sai\_clk\_direction\_t

Enumerator

*kSaiClkInternal* Clock generated internal.

*kSaiClkExternal* Clock generated external.

#### 26.2.2.4 enum sai\_data\_order\_t

Enumerator

*kSaiLSBFirst* Least significant bit transferred first.

*kSaiMSBFirst* Most significant bit transferred first.

#### 26.2.2.5 enum sai\_sync\_mode\_t

Enumerator

*kSaiModeAsync* Asynchronous mode.

*kSaiModeSync* Synchronous mode (with receiver or transmit)

*kSaiModeSyncWithOtherTx* Synchronous with another SAI transmit.

*kSaiModeSyncWithOtherRx* Synchronous with another SAI receiver.

### 26.2.2.6 enum sai\_mclk\_source\_t

Enumerator

*kSaiMclkSourceSysclk* Master clock from the system clock.

*kSaiMclkSourceSelect1* Master clock from source 1.

*kSaiMclkSourceSelect2* Master clock from source 2.

*kSaiMclkSourceSelect3* Master clock from source 3.

### 26.2.2.7 enum sai\_bclk\_source\_t

Enumerator

*kSaiBclkSourceBusclk* Bit clock using bus clock.

*kSaiBclkSourceMclkDiv* Bit clock using master clock divider.

*kSaiBclkSourceOtherSai0* Bit clock from other SAI device.

*kSaiBclkSourceOtherSai1* Bit clock from other SAI device.

### 26.2.2.8 enum sai\_interrupt\_request\_t

Enumerator

*kSaiIntrequestWordStart* Word start flag, means the first word in a frame detected.

*kSaiIntrequestSyncError* Sync error flag, means the sync error is detected.

*kSaiIntrequestFIFOWarning* FIFO warning flag, means the FIFO is empty.

*kSaiIntrequestFIFOError* FIFO error flag.

*kSaiIntrequestFIFORequest* FIFO request, means reached watermark.

### 26.2.2.9 enum sai\_dma\_request\_t

Enumerator

*kSaiDmaReqFIFOWarning* FIFO warning caused by the DMA request.

*kSaiDmaReqFIFORequest* FIFO request caused by the DMA request.

### 26.2.2.10 enum sai\_state\_flag\_t

Enumerator

*kSaiStateFlagWordStart* Word start flag, means the first word in a frame detected.

*kSaiStateFlagSyncError* Sync error flag, means the sync error is detected.

*kSaiStateFlagFIFOError* FIFO error flag.

*kSaiStateFlagSoftReset* Software reset flag.

## SAI HAL driver

### 26.2.2.11 enum sai\_reset\_type\_t

Enumerator

*kSaiResetTypeSoftware* Software reset, reset the logic state.

*kSaiResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.

### 26.2.2.12 enum sai\_run\_mode\_t

Enumerator

*kSaiRunModeDebug* In debug mode.

*kSaiRunModeStop* In stop mode.

## 26.2.3 Function Documentation

### 26.2.3.1 void SAI\_HAL\_TxInit ( uint32\_t saiBaseAddr )

The initialization resets the SAI module by setting the SR bit of TCSR register. Note that the function writes 0 to every control registers.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

### 26.2.3.2 void SAI\_HAL\_RxInit ( uint32\_t saiBaseAddr )

The initialization resets the SAI module by setting the SR bit of RCSR register. Note that the function writes 0 to every control registers.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

### 26.2.3.3 void SAI\_HAL\_TxSetProtocol ( uint32\_t saiBaseAddr, sai\_protocol\_t protocol )

The bus mode means which protocol SAI uses. It can be I2S left, right and so on. Each protocol has a different configuration on bit clock and frame sync.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                                        |
| <i>protocol</i>    | The protocol selection. It can be I2S left aligned, I2S right aligned, etc. |

#### 26.2.3.4 void SAI\_HAL\_RxSetProtocol ( uint32\_t *saiBaseAddr*, sai\_protocol\_t *protocol* )

The bus mode means which protocol SAI uses. It can be I2S left, right and so on. Each protocol has a different configuration on bit clock and frame sync.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                                        |
| <i>protocol</i>    | The protocol selection. It can be I2S left aligned, I2S right aligned, etc. |

#### 26.2.3.5 void SAI\_HAL\_TxSetMasterSlave ( uint32\_t *saiBaseAddr*, sai\_master\_slave\_t *master\_slave\_mode* )

The function determines master or slave mode. Master mode provides its own clock and slave mode uses an external clock.

Parameters

|                          |                                      |
|--------------------------|--------------------------------------|
| <i>saiBaseAddr</i>       | Register base address of SAI module. |
| <i>master_slave_mode</i> | Mater or slave mode.                 |

#### 26.2.3.6 void SAI\_HAL\_RxSetMasterSlave ( uint32\_t *saiBaseAddr*, sai\_master\_slave\_t *master\_slave\_mode* )

The function determines master or slave mode. Master mode provides its own clock and slave mode uses external clock.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

## SAI HAL driver

|                          |                      |
|--------------------------|----------------------|
| <i>master_slave_mode</i> | Mater or slave mode. |
|--------------------------|----------------------|

### 26.2.3.7 static void SAI\_HAL\_SetMclkSrc ( uint32\_t *saiBaseAddr*, sai\_mclk\_source\_t *source* ) [inline], [static]

The source of the clock is different from socs. This function sets the clock source for SAI master clock source. Master clock is used to produce the bit clock for the data transfer.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>source</i>      | Mater clock source                   |

### 26.2.3.8 static uint32\_t SAI\_HAL\_GetMclkSrc ( uint32\_t *saiBaseAddr* ) [inline], [static]

The source of the clock is different from socs. This function gets the clock source for SAI master clock source. Master clock is used to produce the bit clock for the data transfer.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

Returns

Mater clock source

### 26.2.3.9 static void SAI\_HAL\_SetMclkDividerCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]

This function would decides the direction of bit clock generated.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

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

### 26.2.3.10 static bool SAI\_HAL\_GetMclkDivUpdatingCmd ( uint32\_t *saiBaseAddr* ) [inline], [static]

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

Returns

True if the divider updated otherwise false.

### 26.2.3.11 static void SAI\_HAL\_TxSetBclkSrc ( uint32\_t *saiBaseAddr*, sai\_bclk\_source\_t *source* ) [inline], [static]

It is generated by the master clock, bus clock and other devices.

The function sets the source of the bit clock. The bit clock can be produced by the master clock and from the bus clock or other SAI Tx/Rx. Tx and Rx in the SAI module use the same bit clock either from Tx or Rx.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>source</i>      | Bit clock source.                    |

### 26.2.3.12 static void SAI\_HAL\_RxSetBclkSrc ( uint32\_t *saiBaseAddr*, sai\_bclk\_source\_t *source* ) [inline], [static]

It is generated by the master clock, bus clock and other devices.

The function sets the source of the bit clock. The bit clock can be produced by the master clock, and from the bus clock or other SAI Tx/Rx. Tx and Rx in the SAI module use the same bit clock either from Tx or Rx.

Parameters

## SAI HAL driver

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>source</i>      | Bit clock source.                    |

### 26.2.3.13 static uint32\_t SAI\_HAL\_TxGetBclkSrc ( uint32\_t *saiBaseAddr* ) [inline], [static]

It is generated by the master clock, bus clock and other devices.

The function gets the source of the bit clock. The bit clock can be produced by the master clock and from the bus clock or other SAI Tx/Rx. Tx and Rx in the SAI module use the same bit clock either from Tx or Rx.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

Returns

Bit clock source.

### 26.2.3.14 static uint32\_t SAI\_HAL\_RxGetBclkSrc ( uint32\_t *saiBaseAddr* ) [inline], [static]

It is generated by the master clock, bus clock and other devices.

The function gets the source of the bit clock. The bit clock can be produced by the master clock, and from the bus clock or other SAI Tx/Rx. Tx and Rx in the SAI module use the same bit clock either from Tx or Rx.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

Returns

Bit clock source.

### 26.2.3.15 static void SAI\_HAL\_TxSetBclkDiv ( uint32\_t *saiBaseAddr*, uint32\_t *divider* ) [inline], [static]

bclk = mclk / divider. At the same time, bclk = sample\_rate \* channel \* bits. This means how much time is needed to transfer one bit. Notice: The function is called while the bit clock source is the master clock.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>divider</i>     | The divide number of bit clock.      |

#### 26.2.3.16 static void SAI\_HAL\_RxSetBclkDiv ( uint32\_t *saiBaseAddr*, uint32\_t *divider* ) [inline], [static]

bclk = mclk / divider. At the same time, bclk = sample\_rate \* channel \* bits. This means how much time is needed to transfer one bit. Notice: The function is called while the bit clock source is the master clock.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>divider</i>     | The divide number of bit clock.      |

#### 26.2.3.17 static void SAI\_HAL\_TxSetBclkCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]

Parameters

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.    |
| <i>enable</i>      | True means enable, false means disable. |

#### 26.2.3.18 static void SAI\_HAL\_RxSetBclkCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]

Parameters

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.    |
| <i>enable</i>      | True means enable, false means disable. |

#### 26.2.3.19 static void SAI\_HAL\_TxSetBclkInputCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]

## SAI HAL driver

Parameters

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.    |
| <i>enable</i>      | True means enable, false means disable. |

**26.2.3.20 static void SAI\_HAL\_RxSetBclkInputCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.    |
| <i>enable</i>      | True means enable, false means disable. |

**26.2.3.21 static void SAI\_HAL\_TxSetSwapBclkCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]**

This field swaps the bit clock used by the transmitter. When the transmitter is configured in asynchronous mode and this bit is set, the transmitter is clocked by the receiver bit clock. This allows the transmitter and receiver to share the same bit clock, but the transmitter continues to use the transmit frame sync (S-AI\_TX\_SYNC). When the transmitter is configured in synchronous mode, the transmitter BCS field and receiver BCS field must be set to the same value. When both are set, the transmitter and receiver are both clocked by the transmitter bit clock (SAI\_TX\_BCLK) but use the receiver frame sync (SAI\_RX\_SYNC).

Parameters

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.            |
| <i>enable</i>      | True means swap bit closk, false means no swap. |

**26.2.3.22 static void SAI\_HAL\_RxSetSwapBclkCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]**

This field swaps the bit clock used by the receiver. When the receiver is configured in asynchronous mode and this bit is set, the receiver is clocked by the transmitter bit clock (SAI\_RX\_BCLK). This allows the transmitter and receiver to share the same bit clock, but the receiver continues to use the receiver frame sync (SAI\_RX\_SYNC). When the receiver is configured in synchronous mode, the transmitter BCS field and receiver BCS field must be set to the same value. When both are set, the transmitter and receiver are both clocked by the receiver bit clock (SAI\_RX\_BCLK) but use the transmitter frame sync (SAI\_TX\_SYNC).

Parameters

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.            |
| <i>enable</i>      | True means swap bit closk, false means no swap. |

### 26.2.3.23 static void SAI\_HAL\_TxSetBclkDir( uint32\_t *saiBaseAddr*, sai\_clk\_direction\_t *direction* ) [inline], [static]

This function sets the direction of the bit clock generated.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.      |
| <i>direction</i>   | Bit clock generated internal or external. |

### 26.2.3.24 static void SAI\_HAL\_RxSetBclkDir( uint32\_t *saiBaseAddr*, sai\_clk\_direction\_t *direction* ) [inline], [static]

This function sets the direction of the bit clock generated.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.      |
| <i>direction</i>   | Bit clock generated internal or external. |

### 26.2.3.25 static void SAI\_HAL\_TxSetBclkPolarity( uint32\_t *saiBaseAddr*, sai\_clk\_polarity\_t *pol* ) [inline], [static]

Parameters

|                    |                                                                               |
|--------------------|-------------------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                                          |
| <i>pol</i>         | Polarity of the SAI bit clock, which can be configured to active high or low. |

### 26.2.3.26 static void SAI\_HAL\_RxSetBclkPolarity( uint32\_t *saiBaseAddr*, sai\_clk\_polarity\_t *pol* ) [inline], [static]

## SAI HAL driver

Parameters

|                    |                                                                           |
|--------------------|---------------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                                      |
| <i>pol</i>         | Polarity of SAI bit clock, which can be configured to active high or low. |

### 26.2.3.27 static void SAI\_HAL\_TxSetFrameSize ( uint32\_t *saiBaseAddr*, uint32\_t *size* ) [inline], [static]

The frame size means how many words are in a frame. For example 2-channel audio data, the frame size is 2, which means 2 words in a frame.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>size</i>        | Words number in a frame.             |

### 26.2.3.28 static void SAI\_HAL\_RxSetFrameSize ( uint32\_t *saiBaseAddr*, uint32\_t *size* ) [inline], [static]

The frame size means how many words are in a frame. For example 2-channel audio data, the frame size is 2, which means 2 words in a frame.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>size</i>        | Words number in a frame.             |

### 26.2.3.29 static uint32\_t SAI\_HAL\_TxGetFrameSize ( uint32\_t *saiBaseAddr* ) [inline], [static]

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

### 26.2.3.30 static uint32\_t SAI\_HAL\_RxGetFrameSize ( uint32\_t *saiBaseAddr* ) [inline], [static]

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

#### 26.2.3.31 static void SAI\_HAL\_TxSetFrameSyncWidth ( uint32\_t *saiBaseAddr*, uint32\_t *width* ) [inline], [static]

A sync is the number of bit clocks of a frame. The sync width cannot be longer than the length of the first word of the frame.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>width</i>       | How many bit clock in a sync.        |

#### 26.2.3.32 static void SAI\_HAL\_RxSetFrameSyncWidth ( uint32\_t *saiBaseAddr*, uint32\_t *width* ) [inline], [static]

A sync is the number of bit clocks of a frame. The sync width cannot be longer than the length of the first word of the frame.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>width</i>       | How many bit clock in a sync.        |

#### 26.2.3.33 static void SAI\_HAL\_TxSetFrameSyncPolarity ( uint32\_t *saiBaseAddr*, sai\_clk\_polarity\_t *pol* ) [inline], [static]

Parameters

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                                 |
| <i>pol</i>         | Polarity of sai frame sync, can be configured to active high or low. |

#### 26.2.3.34 static void SAI\_HAL\_RxSetFrameSyncPolarity ( uint32\_t *saiBaseAddr*, sai\_clk\_polarity\_t *pol* ) [inline], [static]

## SAI HAL driver

Parameters

|                    |                                                                      |
|--------------------|----------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module..                                |
| <i>pol</i>         | Polarity of SAI frame sync, can be configured to active high or low. |

### 26.2.3.35 static void SAI\_HAL\_TxSetFrameSyncDir ( *uint32\_t saiBaseAddr*, *sai\_clk\_direction\_t direction* ) [inline], [static]

This function sets the direction of frame sync.

Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.       |
| <i>direction</i>   | Frame sync generated internal or external. |

### 26.2.3.36 static void SAI\_HAL\_RxSetFrameSyncDir ( *uint32\_t saiBaseAddr*, *sai\_clk\_direction\_t direction* ) [inline], [static]

This function sets the direction of frame sync.

Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.       |
| <i>direction</i>   | Frame sync generated internal or external. |

### 26.2.3.37 static void SAI\_HAL\_TxSetBitOrder ( *uint32\_t saiBaseAddr*, *sai\_data\_order\_t order* ) [inline], [static]

This function sets the data transfer order. It can be set to MSB first or LSB first.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.      |
| <i>order</i>       | MSB transmit first or LSB transmit first. |

### 26.2.3.38 static void SAI\_HAL\_RxSetBitOrder ( *uint32\_t saiBaseAddr*, *sai\_data\_order\_t order* ) [inline], [static]

This function sets the data transfer order. It can be set to MSB first or LSB first.

Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.      |
| <i>order</i>       | MSB transmit first or LSB transmit first. |

### 26.2.3.39 static void SAI\_HAL\_TxSetFrameSyncEarlyCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]

Parameters

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                              |
| <i>enable</i>      | True means the frame sync one bit early and false means no early. |

### 26.2.3.40 static void SAI\_HAL\_RxSetFrameSyncEarlyCmd ( uint32\_t *saiBaseAddr*, bool *enable* ) [inline], [static]

Parameters

|                    |                                                                   |
|--------------------|-------------------------------------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.                              |
| <i>enable</i>      | True means the frame sync one bit early and false means no early. |

### 26.2.3.41 static void SAI\_HAL\_TxSetWordSize ( uint32\_t *saiBaseAddr*, uint32\_t *bits* ) [inline], [static]

The word size means the quantization level of audio file. SAI supports the 8 bit, 16 bit, 24 bit, and 32 bit formats.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>bits</i>        | How many bits in a word.             |

### 26.2.3.42 static void SAI\_HAL\_RxSetWordSize ( uint32\_t *saiBaseAddr*, uint32\_t *bits* ) [inline], [static]

The word size means the quantization level of audio file. SAI supports 8 bit, 16 bit, 24 bit, and 32 bit formats.

## SAI HAL driver

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>bits</i>        | How many bits in a word.             |

**26.2.3.43 static uint32\_t SAI\_HAL\_TxGetWordSize ( uint32\_t *saiBaseAddr* ) [inline], [static]**

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

**26.2.3.44 static uint32\_t SAI\_HAL\_RxGetWordSize ( uint32\_t *saiBaseAddr* ) [inline], [static]**

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
|--------------------|--------------------------------------|

**26.2.3.45 static void SAI\_HAL\_TxSetFirstWordSize ( uint32\_t *saiBaseAddr*, uint8\_t *size* ) [inline], [static]**

In I2S protocol, the size of the first word is the same as the size of other words. In some protocols, for example, AC'97, the first word is not the same size as others. This function sets the length of the first word which is, in most situations, the same as others.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>size</i>        | The length of frame head word.       |

**26.2.3.46 static void SAI\_HAL\_RxSetFirstWordSize ( uint32\_t *saiBaseAddr*, uint8\_t *size* ) [inline], [static]**

In I2S protocol, the size of the first word is the same as the size of other words. In some protocols, for example, AC'97, the first word is not the same size as others. This function sets the length of the first word which is, in most situations, the same as others.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>size</i>        | The length of frame head word.       |

#### 26.2.3.47 static void SAI\_HAL\_TxSetFirstBitShifted ( uint32\_t *saiBaseAddr*, uint32\_t *index* ) [inline], [static]

The FIFO is 32-bit in SAI. However, not all audio data is 32-bit, but is mostly 16-bit. In this situation, the codec needs to know which bit of the FIFO marks the valid audio data.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>index</i>       | First bit shifted in FIFO.           |

#### 26.2.3.48 static void SAI\_HAL\_RxSetFirstBitShifted ( uint32\_t *saiBaseAddr*, uint32\_t *index* ) [inline], [static]

The FIFO is 32-bit in SAI. However, not all audio data is 32-bit, but is mostly 16-bit. In this situation, the codec needs to know which bit of the FIFO marks the valid audio data.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module. |
| <i>index</i>       | First bit shifted in FIFO.           |

### 26.3 SAI Peripheral driver

#### 26.3.1 Overview

This chapter describes the programming interface of the SAI Peripheral driver. The SAI driver initializes, configures, starts, and stops the SAI. The SAI driver also implements configuration functions, sends, and receives data.

#### 26.3.2 SAI Initialization

To initialize SAI, call the [SAI\\_DRV\\_TxInit\(\)](#) or [SAI\\_DRV\\_RxInit\(\)](#) function and pass the parameters. The parameter is the SAI instance and SAI configuration structure. The function opens the clock gate and initializes the SAI modules according to the structure information.

#### 26.3.3 SAI Configuration

Configuration is implemented by the [SAI\\_DRV\\_TxConfigDataFormat\(\)](#) function. To use this function, transfer the audio data format to the SAI module.

#### 26.3.4 SAI Call diagram

To use the SAI driver, follow these steps, and use Tx as an example:

1. Initialize the SAI module by calling the [SAI\\_DRV\\_TxInit\(\)](#) function.
2. Configure the audio data features of SAI by calling the [SAI\\_DRV\\_TxConfigDataFormat\(\)](#) function.
3. Send/receive data by calling the [SAI\\_DRV\\_SendData\(\)](#) or the [SAI\\_DRV\\_ReceiveData\(\)](#) functions.
4. Start Tx or Rx by calling the [SAI\\_DRV\\_TxStartModule\(\)](#) or the [SAI\\_DRV\\_RxStartModule\(\)](#) function.
5. Shut down the SAI module by calling the [SAI\\_DRV\\_TxDeinit\(\)](#) function.

This is example code to initialize and configure the SAI driver in the DMA mode:

```
// Initialize handler structure.
sai_handler_t handler;
handler.direction = AUDIO_TX;
handler.instance = 0;
handler.fifo_channel = 0;

//Initialize config structure.
sai_user_config_t tx_config;
tx_config.bus_type = kSaiBusI2SLeft;
tx_config.channel = 0;
tx_config.slave_master = kSaiMaster;
tx_config.sync_mode = kSaiModeAsync;
tx_config.bclk_source = kSaiBclkSourceMclkDiv;
tx_config.mclk_source = kSaiMclkSourceSysclk;
tx_config.mclk_divide_enable = true;
tx_config.watermark = 4;
```

```

//SAI state, used for driver internal logic.
sai_state_t tx_state;

//Data format of audio data
audio_data_format_t format;
format.bits = 16;
format.sample_rate = 44100;
format.mclk = 384 * format->sample_rate;
format.words = 2;
//Initialize SAI Tx.
SAI_DRV_TxInit(instance, &tx_config, &tx_state);
//Configure the data format of SAI tx.
SAI_DRV_TxConfigDataFormat(instance,&format);

//option: register callback functions for finished transfer
SAI_DRV_TxRegisterCallback(instance, callback, callback_param);
//start send data
SAI_DRV_SendData(instance, addr, len);
//Enable interrupt
SAI_DRV_TxIntCmd(instance, true);
//Start Tx
SAI_DRV_TxStartModule(instance);

.....

//Stop Tx.
SAI_DRV_TxStopModule(instance);
//Deinit Tx.
SAI_DRV_TxDeinit(instance);

```

## Files

- file [fsl\\_sai\\_driver.h](#)

## Data Structures

- struct [sai\\_data\\_format\\_t](#)  
*Defines the PCM data format.* [More...](#)
- struct [sai\\_state\\_t](#)  
*SAI internal state Users should allocate and transfer memory to the PD during the initialization function.*  
[More...](#)
- struct [sai\\_user\\_config\\_t](#)  
*The description structure for the SAI TX/RX module.* [More...](#)

## Typedefs

- [typedef void\(\\* sai\\_callback\\_t \)\(void \\*parameter\)](#)  
*SAI callback function.*

## Enumerations

- enum [sai\\_status\\_t](#)

## SAI Peripheral driver

*Status structure for SAI.*

## Functions

- **sai\_status\_t SAI\_DRV\_TxInit** (uint32\_t instance, [sai\\_user\\_config\\_t](#) \*config, [sai\\_state\\_t](#) \*state)  
*Initializes the SAI module.*
- **sai\_status\_t SAI\_DRV\_RxInit** (uint32\_t instance, [sai\\_user\\_config\\_t](#) \*config, [sai\\_state\\_t](#) \*state)  
*Initializes the SAI Rx module.*
- **void SAI\_DRV\_TxGetDefaultSetting** ([sai\\_user\\_config\\_t](#) \*config)  
*Gets the default setting of the user configuration.*
- **void SAI\_DRV\_RxGetDefaultSetting** ([sai\\_user\\_config\\_t](#) \*config)  
*Gets the default setting of the user configuration.*
- **sai\_status\_t SAI\_DRV\_TxDeinit** (uint32\_t instance)  
*De-initializes the SAI Tx module.*
- **sai\_status\_t SAI\_DRV\_RxDeinit** (uint32\_t instance)  
*De-initializes the SAI Rx module.*
- **sai\_status\_t SAI\_DRV\_TxConfigDataFormat** (uint32\_t instance, [sai\\_data\\_format\\_t](#) \*format)  
*Configures audio data format of Tx.*
- **sai\_status\_t SAI\_DRV\_RxConfigDataFormat** (uint32\_t instance, [sai\\_data\\_format\\_t](#) \*format)  
*Configures audio data format of Rx.*
- **void SAI\_DRV\_TxStartModule** (uint32\_t instance)  
*Starts the Tx transfer.*
- **void SAI\_DRV\_RxStartModule** (uint32\_t instance)  
*Starts the Rx receive process.*
- **static void SAI\_DRV\_TxStopModule** (uint32\_t instance)  
*Stops writing data to FIFO to disable the DMA or the interrupt request bit.*
- **static void SAI\_DRV\_RxStopModule** (uint32\_t instance)  
*Stops receiving data from FIFO to disable the DMA or the interrupt request bit.*
- **static void SAI\_DRV\_TxSetIntCmd** (uint32\_t instance, bool enable)  
*Enables or disables the Tx interrupt source.*
- **static void SAI\_DRV\_RxSetIntCmd** (uint32\_t instance, bool enable)  
*Enables or disables the Rx interrupt source.*
- **static void SAI\_DRV\_TxSetDmaCmd** (uint32\_t instance, bool enable)  
*Enables or disables the Tx DMA source.*
- **static void SAI\_DRV\_RxSetDmaCmd** (uint32\_t instance, bool enable)  
*Enables or disables the Rx interrupt source.*
- **static uint32\_t \* SAI\_DRV\_TxGetFifoAddr** (uint32\_t instance, uint32\_t fifo\_channel)  
*Gets the Tx FIFO address of the data channel.*
- **static uint32\_t \* SAI\_DRV\_RxGetFifoAddr** (uint32\_t instance, uint32\_t fifo\_channel)  
*Gets the Rx FIFO address of the data channel.*
- **uint32\_t SAI\_DRV\_SendData** (uint32\_t instance, uint8\_t \*addr, uint32\_t len)  
*Sends data of a certain length.*
- **uint32\_t SAI\_DRV\_ReceiveData** (uint32\_t instance, uint8\_t \*addr, uint32\_t len)  
*Receives data a certain length.*
- **uint32\_t SAI\_DRV\_SendDataBlocking** (uint32\_t instance, uint8\_t \*addr, uint32\_t len)  
*Sends data of a certain length in a blocking way.*
- **uint32\_t SAI\_DRV\_ReceiveDataBlocking** (uint32\_t instance, uint8\_t \*addr, uint32\_t len)  
*Receives data of a certain length in a blocking way.*
- **void SAI\_DRV\_TxRegisterCallback** (uint32\_t instance, [sai\\_callback\\_t](#) callback, void \*callback\_param)

- Registers the callback function after completing a send.
- void **SAI\_DRV\_RxRegisterCallback** (uint32\_t instance, **sai\_callback\_t** callback, void \*callback\_param)
  - Registers the callback function after completing a receive.
- void **SAI\_DRV\_TxIRQHandler** (uint32\_t instance)
  - Default SAI Tx interrupt handler.
- void **SAI\_DRV\_RxIRQHandler** (uint32\_t instance)
  - Default SAI Rx interrupt handler.

## 26.3.5 Data Structure Documentation

### 26.3.5.1 struct sai\_data\_format\_t

#### Data Fields

- uint32\_t **sample\_rate**  
*Sample rate of the PCM file.*
- uint32\_t **mclk**  
*Master clock frequency.*
- uint8\_t **bits**  
*How many bits in a word.*
- **sai\_mono\_streo\_t mono\_streo**  
*How many word in a frame.*

### 26.3.5.2 struct sai\_state\_t

Note: During the SAI execution, users should not free the state. Otherwise, the driver malfunctions.

### 26.3.5.3 struct sai\_user\_config\_t

#### Data Fields

- **sai\_mclk\_source\_t mclk\_source**  
*Master clock source.*
- bool **mclk\_divide\_enable**  
*Enable the divide of master clock to generate bit clock.*
- **sai\_sync\_mode\_t sync\_mode**  
*Synchronous or asynchronous.*
- **sai\_protocol\_t protocol**  
*I2S left, I2S right or I2S type.*
- **sai\_master\_slave\_t slave\_master**  
*Master or slave.*
- **sai\_bclk\_source\_t bclk\_source**  
*Bit clock from master clock or other modules.*
- uint8\_t **channel**  
*Which FIFO is used to transfer.*

## SAI Peripheral driver

### 26.3.5.3.0.53 Field Documentation

26.3.5.3.0.53.1 `sai_mclk_source_t sai_user_config_t::mclk_source`

26.3.5.3.0.53.2 `bool sai_user_config_t::mclk_divide_enable`

26.3.5.3.0.53.3 `sai_sync_mode_t sai_user_config_t::sync_mode`

26.3.5.3.0.53.4 `sai_protocol_t sai_user_config_t::protocol`

26.3.5.3.0.53.5 `sai_master_slave_t sai_user_config_t::slave_master`

26.3.5.3.0.53.6 `sai_bclk_source_t sai_user_config_t::bclk_source`

26.3.5.3.0.53.7 `uint8_t sai_user_config_t::channel`

### 26.3.6 Function Documentation

26.3.6.1 `sai_status_t SAI_DRV_TxInit ( uint32_t instance, sai_user_config_t * config, sai_state_t * state )`

This function initializes the SAI registers according to the configuration structure. This function also initializes the basic SAI settings including board-relevant settings. Notice: This function does not initialize an entire SAI instance. It only initializes the Tx according to the value in the handler.

Parameters

|                       |                                     |
|-----------------------|-------------------------------------|
| <code>instance</code> | SAI module instance.                |
| <code>config</code>   | The configuration structure of SAI. |
| <code>state</code>    | Pointer of SAI run state structure. |

Returns

Return kStatus\_SAI\_Success while the initialize success and kStatus\_SAI\_Fail if failed.

26.3.6.2 `sai_status_t SAI_DRV_RxInit ( uint32_t instance, sai_user_config_t * config, sai_state_t * state )`

This function initializes the SAI registers according to the configuration structure. This function also initializes the basic SAI settings including board-relevant settings. Note that this function does not initialize an entire SAI instance. This function only initializes the Rx according to the value in the handler.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | SAI module instance.                |
| <i>config</i>   | The configuration structure of SAI. |
| <i>state</i>    | Pointer of SAI run state structure. |

Returns

Return kStatus\_SAI\_Success while the initialize success and kStatus\_SAI\_Fail if failed.

### 26.3.6.3 void SAI\_DRV\_TxGetDefaultSetting ( *sai\_user\_config\_t \* config* )

The default settings for SAI are:

- Audio protocol is I2S format
- Watermark is 4
- Use SAI0
- Channel is channel0
- SAI as master
- MCLK from system core clock
- Tx is in an asynchronous mode

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | Pointer of user configure structure. |
|---------------|--------------------------------------|

### 26.3.6.4 void SAI\_DRV\_RxGetDefaultSetting ( *sai\_user\_config\_t \* config* )

The default settings for SAI are:

- Audio protocol is I2S format
- Watermark is 4
- Use SAI0
- Data channel is channel0
- SAI as master
- MCLK from system core clock
- Rx is in synchronous way

Parameters

## SAI Peripheral driver

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | Pointer of user configure structure. |
|---------------|--------------------------------------|

### 26.3.6.5 **sai\_status\_t SAI\_DRV\_TxDeinit ( uint32\_t *instance* )**

This function closes the SAI Tx device. It does not close the entire SAI instance, however. It only closes the clock gate while both Tx and Rx are closed in the same instance.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

Returns

Return kStatus\_SAI\_Success while the process success and kStatus\_SAI\_Fail if failed.

### 26.3.6.6 **sai\_status\_t SAI\_DRV\_RxDeinit ( uint32\_t *instance* )**

This function closes the SAI Rx device. It does not close the entire SAI instance, however. It only closes the clock gate while both Tx and Rx are closed in the same instance.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

Returns

Return kStatus\_SAI\_Success while the process success and kStatus\_SAI\_Fail if failed.

### 26.3.6.7 **sai\_status\_t SAI\_DRV\_TxConfigDataFormat ( uint32\_t *instance*, sai\_data\_format\_t \* *format* )**

The function configures an audio sample rate, data bits, and a channel number.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

|               |                                    |
|---------------|------------------------------------|
| <i>format</i> | PCM data format structure pointer. |
|---------------|------------------------------------|

Returns

Return kStatus\_SAI\_Success while the process success and kStatus\_SAI\_Fail if failed.

#### 26.3.6.8 **sai\_status\_t SAI\_DRV\_RxConfigDataFormat ( uint32\_t *instance*, sai\_data\_format\_t \* *format* )**

The function configures an audio sample rate, data bits, and a channel number.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>instance</i> | SAI module instance of the SAI module. |
| <i>format</i>   | PCM data format structure pointer.     |

Returns

Return kStatus\_SAI\_Success while the process success and kStatus\_SAI\_Fail if failed.

#### 26.3.6.9 **void SAI\_DRV\_TxStartModule ( uint32\_t *instance* )**

The function enables the interrupt/DMA request source and the transmit channel.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

#### 26.3.6.10 **void SAI\_DRV\_RxStartModule ( uint32\_t *instance* )**

The function enables the interrupt/DMA request source and the transmit channel.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>instance</i> | SAI module instance of the SAI module. |
|-----------------|----------------------------------------|

#### 26.3.6.11 **static void SAI\_DRV\_TxStopModule ( uint32\_t *instance* ) [inline], [static]**

This function provides the method to pause writing data.

## SAI Peripheral driver

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

**26.3.6.12 static void SAI\_DRV\_RxStopModule ( uint32\_t *instance* ) [inline], [static]**

This function provides the method to pause writing data.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

**26.3.6.13 static void SAI\_DRV\_TxSetIntCmd ( uint32\_t *instance*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <i>instance</i> | SAI module instance.                                                      |
| <i>enable</i>   | True means enable interrupt source, false means disable interrupt source. |

**26.3.6.14 static void SAI\_DRV\_RxSetIntCmd ( uint32\_t *instance*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <i>instance</i> | SAI module instance.                                                      |
| <i>enable</i>   | True means enable interrupt source, false means disable interrupt source. |

**26.3.6.15 static void SAI\_DRV\_TxSetDmaCmd ( uint32\_t *instance*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| <i>instance</i> | SAI module instance.                                          |
| <i>enable</i>   | True means enable DMA source, false means disable DMA source. |

**26.3.6.16 static void SAI\_DRV\_RxSetDmaCmd ( uint32\_t *instance*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| <i>instance</i> | SAI module instance.                                          |
| <i>enable</i>   | True means enable DMA source, false means disable DMA source. |

**26.3.6.17 static uint32\_t\* SAI\_DRV\_TxGetFifoAddr ( uint32\_t *instance*, uint32\_t *fifo\_channel* ) [inline], [static]**

This function is mainly used for the DMA settings, which the DMA configuration needs for the SAI source/destination address.

Parameters

|                     |                                        |
|---------------------|----------------------------------------|
| <i>instance</i>     | SAI module instance of the SAI module. |
| <i>fifo_channel</i> | FIFO channel of SAI Tx.                |

Returns

Returns the address of the data channel FIFO.

**26.3.6.18 static uint32\_t\* SAI\_DRV\_RxGetFifoAddr ( uint32\_t *instance*, uint32\_t *fifo\_channel* ) [inline], [static]**

This function is mainly used for the DMA settings, which the DMA configuration needs for the SAI source/destination address.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>instance</i> | SAI module instance of the SAI module. |
|-----------------|----------------------------------------|

## SAI Peripheral driver

|                     |                         |
|---------------------|-------------------------|
| <i>fifo_channel</i> | FIFO channel of SAI Rx. |
|---------------------|-------------------------|

Returns

Returns the address of the data channel FIFO.

### 26.3.6.19 `uint32_t SAI_DRV_SendData ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function sends the data to the Tx FIFO. This function starts the transfer, and, while finishing the transfer, calls the callback function registered by users.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | SAI module instance of the SAI module.             |
| <i>addr</i>     | Address of the data which needs to be transferred. |
| <i>len</i>      | The data length which need to be sent.             |

Returns

Returns the length which was sent.

### 26.3.6.20 `uint32_t SAI_DRV_ReceiveData ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function receives the data from the RX FIFO. This function starts the transfer, and, while finishing the transfer, calls the callback function registered by the user.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | SAI module instance.                               |
| <i>addr</i>     | Address of the data which needs to be transferred. |
| <i>len</i>      | The data length which needs to be received.        |

Returns

Returns the length received.

### 26.3.6.21 `uint32_t SAI_DRV_SendDataBlocking ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function sends the data to the Tx FIFO, does not exit until the data is sent to FIFO, and uses the polling way to send audio data.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | SAI module instance.                               |
| <i>addr</i>     | Address of the data which needs to be transferred. |
| <i>len</i>      | The data length which need to be sent.             |

Returns

Returns the length which was sent.

#### 26.3.6.22 **uint32\_t SAI\_DRV\_ReceiveDataBlocking ( uint32\_t *instance*, uint8\_t \* *addr*, uint32\_t *len* )**

This function receives data from the Rx FIFO, does not exit until the data is received from FIFO, and uses the polling way to send audio data.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | SAI module instance.                               |
| <i>addr</i>     | Address of the data which needs to be transferred. |
| <i>len</i>      | The data length which need to be sent.             |

Returns

Returns the length which was sent.

#### 26.3.6.23 **void SAI\_DRV\_TxRegisterCallback ( uint32\_t *instance*, sai\_callback\_t *callback*, void \* *callback\_param* )**

This function tells SAI which function needs to be called after a period length sending. This callback function is used for non-blocking sending.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | SAI module instance.                |
| <i>callback</i> | Callback function defined by users. |

## SAI Peripheral driver

|                       |                                         |
|-----------------------|-----------------------------------------|
| <i>callback_param</i> | The parameter of the callback function. |
|-----------------------|-----------------------------------------|

### **26.3.6.24 void SAI\_DRV\_RxRegisterCallback ( uint32\_t *instance*, sai\_callback\_t *callback*, void \* *callback\_param* )**

This function tells SAI which function needs to be called after a period length receive. This callback function is used for non-blocking receiving.

Parameters

|                       |                                         |
|-----------------------|-----------------------------------------|
| <i>instance</i>       | SAI module instance.                    |
| <i>callback</i>       | Callback function defined by users.     |
| <i>callback_param</i> | The parameter of the callback function. |

### **26.3.6.25 void SAI\_DRV\_TxIRQHandler ( uint32\_t *instance* )**

This function sends data in the interrupt and checks the FIFO error.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

### **26.3.6.26 void SAI\_DRV\_RxIRQHandler ( uint32\_t *instance* )**

This function receives data in the interrupt and checks the FIFO error.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

# **Chapter 27**

## **Secured Digital Host Controller (SDHC)**

### **27.1 Overview**

The Kinetis SDK provides both HAL and Peripheral drivers for the Secure Digital Host Controller (SDHC) block of Kinetis devices.

### **Modules**

- [SD Card SPI Data Definition](#)
- [SD Card SPI Driver](#)
- [SDHC Card Definition](#)
- [SDHC Card Driver](#)
- [SDHC Card Related Standard Definition](#)
- [SDHC Data Types](#)
- [SDHC HAL](#)
- [SDHC Peripheral Driver](#)
- [SDHC Standard Definition](#)

### 27.2 SDHC HAL

#### 27.2.1 Overview

This section describes the programming interface of the SDHC HAL driver.

## SDHC HAL FUNCTION

- static void [SDHC\\_HAL\\_SetDmaAddress](#) (uint32\_t baseAddr, uint32\_t address)  
*Configures the DMA address.*
- static uint32\_t [SDHC\\_HAL\\_GetDmaAddress](#) (uint32\_t baseAddr)  
*Gets the DMA address.*
- static uint32\_t [SDHC\\_HAL\\_GetBlockSize](#) (uint32\_t baseAddr)  
*Gets the block size configured.*
- static void [SDHC\\_HAL\\_SetBlockSize](#) (uint32\_t baseAddr, uint32\_t blockSize)  
*Sets the block size.*
- static void [SDHC\\_HAL\\_SetBlockCount](#) (uint32\_t baseAddr, uint32\_t blockCount)  
*Sets the block count.*
- static uint32\_t [SDHC\\_HAL\\_GetBlockCount](#) (uint32\_t baseAddr)  
*Gets the block count configured.*
- static void [SDHC\\_HAL\\_SetCmdArgument](#) (uint32\_t baseAddr, uint32\_t arg)  
*Configures the command argument.*
- static void [SDHC\\_HAL\\_SendCmd](#) (uint32\_t baseAddr, uint32\_t index, uint32\_t flags)  
*Sends a command.*
- static void [SDHC\\_HAL\\_SetData](#) (uint32\_t baseAddr, uint32\_t data)  
*Fills the the data port.*
- static uint32\_t [SDHC\\_HAL\\_GetData](#) (uint32\_t baseAddr)  
*Retrieves the data from the data port.*
- static uint32\_t [SDHC\\_HAL\\_IsCmdInhibit](#) (uint32\_t baseAddr)  
*Checks whether the command inhibit bit is set or not.*
- static uint32\_t [SDHC\\_HAL\\_IsDataInhibit](#) (uint32\_t baseAddr)  
*Checks whether data inhibit bit is set or not.*
- static uint32\_t [SDHC\\_HAL\\_IsDataLineActive](#) (uint32\_t baseAddr)  
*Checks whether data line is active.*
- static uint32\_t [SDHC\\_HAL\\_IsSdClockStable](#) (uint32\_t baseAddr)  
*Checks whether the SD clock is stable or not.*
- static uint32\_t [SDHC\\_HAL\\_IsIpgClockOff](#) (uint32\_t baseAddr)  
*Checks whether the IPG clock is off or not.*
- static uint32\_t [SDHC\\_HAL\\_IsSysClockOff](#) (uint32\_t baseAddr)  
*Checks whether the system clock is off or not.*
- static uint32\_t [SDHC\\_HAL\\_IsPeripheralClockOff](#) (uint32\_t baseAddr)  
*Checks whether the peripheral clock is off or not.*
- static uint32\_t [SDHC\\_HAL\\_IsSdClkOff](#) (uint32\_t baseAddr)  
*Checks whether the SD clock is off or not.*
- static uint32\_t [SDHC\\_HAL\\_IsWriteTransferActive](#) (uint32\_t baseAddr)  
*Checks whether the write transfer is active or not.*
- static uint32\_t [SDHC\\_HAL\\_IsReadTransferActive](#) (uint32\_t baseAddr)  
*Checks whether the read transfer is active or not.*
- static uint32\_t [SDHC\\_HAL\\_IsBuffWriteEnabled](#) (uint32\_t baseAddr)  
*Check whether the buffer write is enabled or not.*

- static uint32\_t **SDHC\_HAL\_IsBuffReadEnabled** (uint32\_t baseAddr)  
*Checks whether the buffer read is enabled or not.*
- static uint32\_t **SDHC\_HAL\_IsCardInserted** (uint32\_t baseAddr)  
*Checks whether the card is inserted or not.*
- static uint32\_t **SDHC\_HAL\_IsCmdLineLevelHigh** (uint32\_t baseAddr)  
*Checks whether the command line signal is high or not.*
- static uint32\_t **SDHC\_HAL\_GetDataLineLevel** (uint32\_t baseAddr)  
*Gets the data line signal level or not.*
- static void **SDHC\_HAL\_SetLedState** (uint32\_t baseAddr, sdhc\_hal\_led\_t state)  
*Sets the LED state.*
- static void **SDHC\_HAL\_SetDataTransferWidth** (uint32\_t baseAddr, sdhc\_hal\_dtw\_t dtw)  
*Sets the data transfer width.*
- static bool **SDHC\_HAL\_IsD3cdEnabled** (uint32\_t baseAddr)  
*Checks whether the DAT3 is taken as card detect pin.*
- static void **SDHC\_HAL\_SetD3cd** (uint32\_t baseAddr, bool enable)  
*Enables the DAT3 as a card detect pin.*
- static void **SDHC\_HAL\_SetEndian** (uint32\_t baseAddr, sdhc\_hal\_endian\_t endianMode)  
*Configures the endian mode.*
- static uint32\_t **SDHC\_HAL\_GetCdTestLevel** (uint32\_t baseAddr)  
*Gets the card detect test level.*
- static void **SDHC\_HAL\_SetCdTest** (uint32\_t baseAddr, bool enable)  
*Enables the card detect test.*
- static void **SDHC\_HAL\_SetDmaMode** (uint32\_t baseAddr, sdhc\_hal\_dma\_mode\_t dmaMode)  
*Sets the DMA mode.*
- static void **SDHC\_HAL\_SetStopAtBlockGap** (uint32\_t baseAddr, bool enable)  
*Enables stop at the block gap.*
- static void **SDHC\_HAL\_SetContinueRequest** (uint32\_t baseAddr)  
*Restarts a transaction which has stopped at the block gap.*
- static void **SDHC\_HAL\_SetReadWaitCtrl** (uint32\_t baseAddr, bool enable)  
*Enables the read wait control for the SDIO cards.*
- static void **SDHC\_HAL\_SetIntStopAtBlockGap** (uint32\_t baseAddr, bool enable)  
*Enables stop at the block gap requests.*
- static void **SDHC\_HAL\_SetWakeupOnCardInt** (uint32\_t baseAddr, bool enable)  
*Enables wakeup event on the card interrupt.*
- static void **SDHC\_HAL\_SetWakeupOnCardInsertion** (uint32\_t baseAddr, bool enable)  
*Enables wakeup event on the card insertion.*
- static void **SDHC\_HAL\_SetWakeupOnCardRemoval** (uint32\_t baseAddr, bool enable)  
*Enables wakeup event on card removal.*
- static void **SDHC\_HAL\_SetIpClock** (uint32\_t baseAddr, bool enable)  
*Enables the IPG clock and no automatic clock gating off.*
- static void **SDHC\_HAL\_SetSysClock** (uint32\_t baseAddr, bool enable)  
*Enables the system clock and no automatic clock gating off.*
- static void **SDHC\_HAL\_SetPeripheralClock** (uint32\_t baseAddr, bool enable)  
*Enables the peripheral clock and no automatic clock gating off.*
- static void **SDHC\_HAL\_SetSdClock** (uint32\_t baseAddr, bool enable)  
*Enables the SD clock.*
- static void **SDHC\_HAL\_SetClockDivisor** (uint32\_t baseAddr, uint32\_t divisor)  
*Sets the SD clock frequency divisor.*
- static void **SDHC\_HAL\_SetClockFrequency** (uint32\_t baseAddr, uint32\_t frequency)  
*Sets the SD clock frequency select.*
- static void **SDHC\_HAL\_SetDataTimeout** (uint32\_t baseAddr, uint32\_t timeout)

## SDHC HAL

- static uint32\_t **SDHC\_HAL\_GetIntFlags** (uint32\_t baseAddr)  
*Gets the current interrupt status.*
- static void **SDHC\_HAL\_ClearIntFlags** (uint32\_t baseAddr, uint32\_t mask)  
*Clears a specified interrupt status.*
- static uint32\_t **SDHC\_HAL\_GetIntSignal** (uint32\_t baseAddr)  
*Gets the currently enabled interrupt signal.*
- static uint32\_t **SDHC\_HAL\_GetIntState** (uint32\_t baseAddr)  
*Gets the currently enabled interrupt state.*
- static uint32\_t **SDHC\_HAL\_GetAc12Error** (uint32\_t baseAddr)  
*Gets the auto cmd12 error.*
- static uint32\_t **SDHC\_HAL\_GetMaxBlockLength** (uint32\_t baseAddr)  
*Gets the maximum block length supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportAdma** (uint32\_t baseAddr)  
*Checks whether the ADMA is supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportHighspeed** (uint32\_t baseAddr)  
*Checks whether the high speed is supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportDma** (uint32\_t baseAddr)  
*Checks whether the DMA is supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportSuspendResume** (uint32\_t baseAddr)  
*Checks whether the suspend/resume is supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportV330** (uint32\_t baseAddr)  
*Checks whether the voltage 3.3 is supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportV300** (uint32\_t baseAddr)  
*Checks whether the voltage 3.0 is supported.*
- static uint32\_t **SDHC\_HAL\_DoesHostSupportV180** (uint32\_t baseAddr)  
*Checks whether the voltage 1.8 is supported.*
- static void **SDHC\_HAL\_SetWriteWatermarkLevel** (uint32\_t baseAddr, uint32\_t watermark)  
*Sets the watermark for writing.*
- static void **SDHC\_HAL\_SetReadWatermarkLevel** (uint32\_t baseAddr, uint32\_t watermark)  
*Sets the watermark for reading.*
- static void **SDHC\_HAL\_SetForceEventFlags** (uint32\_t baseAddr, uint32\_t mask)  
*Sets the force events according to the given mask.*
- static uint32\_t **SDHC\_HAL\_IsAdmaLengthMismatchError** (uint32\_t baseAddr)  
*Checks whether the ADMA error is length mismatch.*
- static bool **SDHC\_HAL\_IsSdClockOff** (uint32\_t baseAddr)  
*Checks the SD clock.*
- static uint32\_t **SDHC\_HAL\_GetAdmaErrorState** (uint32\_t baseAddr)  
*Returns the state of the ADMA error.*
- static uint32\_t **SDHC\_HAL\_IsAdmaDescriptorError** (uint32\_t baseAddr)  
*Checks whether the ADMA error is a descriptor error.*
- static void **SDHC\_HAL\_SetAdmaAddress** (uint32\_t baseAddr, uint32\_t address)  
*Sets the ADMA address.*
- static void **SDHC\_HAL\_SetExternalDmaRequest** (uint32\_t baseAddr, bool enable)  
*Enables the external DMA request.*
- static void **SDHC\_HAL\_SetExactBlockNumber** (uint32\_t baseAddr, bool enable)  
*Enables the exact block number for the SDIO CMD53.*
- static void **SDHC\_HAL\_SetBootAckTimeout** (uint32\_t baseAddr, uint32\_t timeout)  
*Sets the timeout value for the boot ACK.*
- static void **SDHC\_HAL\_SetBootAck** (uint32\_t baseAddr, bool enable)  
*Enables the boot ACK.*

- static void **SDHC\_HAL\_SetBootMode** (uint32\_t baseAddr, sdhc\_hal\_mmcboot\_t mode)  
*Configures the boot mode.*
- static void **SDHC\_HAL\_SetFastboot** (uint32\_t baseAddr, bool enable)  
*Enables the fast boot.*
- static void **SDHC\_HAL\_SetAutoStopAtBlockGap** (uint32\_t baseAddr, bool enable)  
*Enables the automatic stop at the block gap.*
- static void **SDHC\_HAL\_SetBootBlockCount** (uint32\_t baseAddr, uint32\_t blockCount)  
*Configures the the block count for the boot.*
- static uint32\_t **SDHC\_HAL\_GetSpecificationVersion** (uint32\_t baseAddr)  
*Gets a specification version.*
- static uint32\_t **SDHC\_HAL\_GetVendorVersion** (uint32\_t baseAddr)  
*Gets the vendor version.*
- uint32\_t **SDHC\_HAL.GetResponse** (uint32\_t baseAddr, uint32\_t index)  
*Gets the command response.*
- void **SDHC\_HAL\_SetIntSignal** (uint32\_t baseAddr, bool enable, uint32\_t mask)  
*Enables the specified interrupts.*
- void **SDHC\_HAL\_SetIntState** (uint32\_t baseAddr, bool enable, uint32\_t mask)  
*Enables the specified interrupt state.*
- uint32\_t **SDHC\_HAL\_Reset** (uint32\_t baseAddr, uint32\_t type, uint32\_t timeout)  
*Performs an SDHC reset.*
- uint32\_t **SDHC\_HAL\_InitCard** (uint32\_t baseAddr, uint32\_t timeout)  
*Sends 80 clocks to the card to initialize the card.*
- IRQn\_Type **SDHC\_HAL\_GetIrqId** (uint32\_t baseAddr)  
*Gets the IRQ ID for a given host controller.*
- void **SDHC\_HAL\_Init** (uint32\_t baseAddr)  
*Initializes the SDHC HAL.*

## 27.2.2 Function Documentation

### 27.2.2.1 static void SDHC\_HAL\_SetDmaAddress ( uint32\_t *baseAddr*, uint32\_t *address* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>address</i>  | the DMA address   |

### 27.2.2.2 static uint32\_t SDHC\_HAL\_GetDmaAddress ( uint32\_t *baseAddr* ) [inline], [static]

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

the DMA address

**27.2.2.3 static uint32\_t SDHC\_HAL\_GetBlockSize ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

the block size already configured

**27.2.2.4 static void SDHC\_HAL\_SetBlockSize ( uint32\_t *baseAddr*, uint32\_t *blockSize* ) [inline], [static]**

Parameters

|                  |                   |
|------------------|-------------------|
| <i>baseAddr</i>  | SDHC base address |
| <i>blockSize</i> | the block size    |

**27.2.2.5 static void SDHC\_HAL\_SetBlockCount ( uint32\_t *baseAddr*, uint32\_t *blockCount* ) [inline], [static]**

Parameters

|                   |                   |
|-------------------|-------------------|
| <i>baseAddr</i>   | SDHC base address |
| <i>blockCount</i> | the block count   |

**27.2.2.6 static uint32\_t SDHC\_HAL\_GetBlockCount ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

the block count already configured

#### 27.2.2.7 static void SDHC\_HAL\_SetCmdArgument ( *uint32\_t baseAddr, uint32\_t arg* ) [inline], [static]

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | SDHC base address    |
| <i>arg</i>      | the command argument |

#### 27.2.2.8 static void SDHC\_HAL\_SendCmd ( *uint32\_t baseAddr, uint32\_t index, uint32\_t flags* ) [inline], [static]

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>baseAddr</i> | SDHC base address   |
| <i>index</i>    | command index       |
| <i>flags</i>    | transfer type flags |

#### 27.2.2.9 static void SDHC\_HAL\_SetData ( *uint32\_t baseAddr, uint32\_t data* ) [inline], [static]

Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>baseAddr</i> | SDHC base address         |
| <i>data</i>     | the data about to be sent |

#### 27.2.2.10 static uint32\_t SDHC\_HAL\_GetData ( *uint32\_t baseAddr* ) [inline], [static]

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

data the data read

### 27.2.2.11 static uint32\_t SDHC\_HAL\_IsCmdInhibit ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if command inhibit, 0 if not.

### 27.2.2.12 static uint32\_t SDHC\_HAL\_IsDataInhibit ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if data inhibit, 0 if not.

### 27.2.2.13 static uint32\_t SDHC\_HAL\_IsDataLineActive ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's active, 0 if not.

#### 27.2.2.14 static uint32\_t SDHC\_HAL\_IsSdClockStable( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's stable, 0 if not.

#### 27.2.2.15 static uint32\_t SDHC\_HAL\_IsLpClockOff( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's off, 0 if not.

#### 27.2.2.16 static uint32\_t SDHC\_HAL\_IsSysClockOff( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's off, 0 if not.

#### 27.2.2.17 static uint32\_t SDHC\_HAL\_IsPeripheralClockOff( uint32\_t *baseAddr* ) [inline], [static]

## SDHC HAL

Parameters

|                 |                    |
|-----------------|--------------------|
| <i>baseAddr</i> | SDHC base address. |
|-----------------|--------------------|

Returns

1 if it's off, 0 if not.

### 27.2.2.18 static uint32\_t SDHC\_HAL\_IsSdClkOff ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's off, 0 if not.

### 27.2.2.19 static uint32\_t SDHC\_HAL\_IsWriteTransferActive ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's active, 0 if not.

### 27.2.2.20 static uint32\_t SDHC\_HAL\_IsReadTransferActive ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's off, 0 if not.

27.2.2.21 **static uint32\_t SDHC\_HAL\_IsBuffWriteEnabled ( uint32\_t *baseAddr* )**  
[**inline**], [**static**]

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's isEnableddd, 0 if not.

**27.2.2.22 static uint32\_t SDHC\_HAL\_IsBuffReadEnabled ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's isEnableddd, 0 if not.

**27.2.2.23 static uint32\_t SDHC\_HAL\_IsCardInserted ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                    |
|-----------------|--------------------|
| <i>baseAddr</i> | SDHC base address. |
|-----------------|--------------------|

Returns

1 if it's inserted, 0 if not.

**27.2.2.24 static uint32\_t SDHC\_HAL\_IsCmdLineLevelHigh ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

1 if it's high, 0 if not.

27.2.2.25 **static uint32\_t SDHC\_HAL\_GetDataLineLevel ( uint32\_t *baseAddr* )**  
[**inline**], [**static**]

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

[7:0] data line signal level

**27.2.2.26 static void SDHC\_HAL\_SetLedState ( uint32\_t *baseAddr*, sdhc\_hal\_led\_t *state* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>state</i>    | the LED state     |

**27.2.2.27 static void SDHC\_HAL\_SetDataTransferWidth ( uint32\_t *baseAddr*, sdhc\_hal\_dtw\_t *dtw* ) [inline], [static]**

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>baseAddr</i> | SDHC base address   |
| <i>dtw</i>      | data transfer width |

**27.2.2.28 static bool SDHC\_HAL\_IsD3cdEnabled ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if DAT3 as card detect pin is enabled

**27.2.2.29 static void SDHC\_HAL\_SetD3cd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | SDHC base address                 |
| <i>enable</i>   | to enable DAT3 as card detect pin |

**27.2.2.30 static void SDHC\_HAL\_SetEndian ( uint32\_t *baseAddr*, sdhc\_hal\_endian\_t *endianMode* ) [inline], [static]**

Parameters

|                   |                   |
|-------------------|-------------------|
| <i>baseAddr</i>   | SDHC base address |
| <i>endianMode</i> | endian mode       |

**27.2.2.31 static uint32\_t SDHC\_HAL\_GetCdTestLevel ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

card detect test level

**27.2.2.32 static void SDHC\_HAL\_SetCdTest ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>baseAddr</i> | SDHC base address                             |
| <i>enable</i>   | to enable card detect signal for test purpose |

**27.2.2.33 static void SDHC\_HAL\_SetDmaMode ( uint32\_t *baseAddr*, sdhc\_hal\_dma\_mode\_t *dmaMode* ) [inline], [static]**

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>dmaMode</i>  | the DMA mode      |

**27.2.2.34 static void SDHC\_HAL\_SetStopAtBlockGap ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>baseAddr</i> | SDHC base address            |
| <i>enable</i>   | to stop at block gap request |

**27.2.2.35 static void SDHC\_HAL\_SetContinueRequest ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

**27.2.2.36 static void SDHC\_HAL\_SetReadWaitCtrl ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>baseAddr</i> | SDHC base address           |
| <i>enable</i>   | to enable read wait control |

**27.2.2.37 static void SDHC\_HAL\_SetIntStopAtBlockGap ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | SDHC base address                |
| <i>enable</i>   | to enable interrupt at block gap |

27.2.2.38 **static void SDHC\_HAL\_SetWakeupOnCardInt ( uint32\_t baseAddr, bool enable ) [inline], [static]**

## SDHC HAL

Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>baseAddr</i> | SDHC base address                        |
| <i>enable</i>   | to enable wakeup event on card interrupt |

**27.2.2.39 static void SDHC\_HAL\_SetWakeupOnCardInsertion ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>baseAddr</i> | SDHC base address                        |
| <i>enable</i>   | to enable wakeup event on card insertion |

**27.2.2.40 static void SDHC\_HAL\_SetWakeupOnCardRemoval ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | SDHC base address                      |
| <i>enable</i>   | to enable wakeup event on card removal |

**27.2.2.41 static void SDHC\_HAL\_SetIpClock ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>baseAddr</i> | SDHC base address   |
| <i>enable</i>   | to enable IPG clock |

**27.2.2.42 static void SDHC\_HAL\_SetSysClock ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>baseAddr</i> | SDHC base address   |
| <i>enable</i>   | to enable SYS clock |

27.2.2.43 **static void SDHC\_HAL\_SetPeripheralClock ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

## SDHC HAL

Parameters

|                 |                            |
|-----------------|----------------------------|
| <i>baseAddr</i> | SDHC base address          |
| <i>enable</i>   | to enable Peripheral clock |

**27.2.2.44 static void SDHC\_HAL\_SetSdClock ( uint32\_t *baseAddr*, bool *enable* )  
[inline], [static]**

It should be disabled before changing the SD clock frequency.

Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>baseAddr</i> | SDHC base address         |
| <i>enable</i>   | to enable SD clock or not |

**27.2.2.45 static void SDHC\_HAL\_SetClockDivisor ( uint32\_t *baseAddr*, uint32\_t *divisor* )  
[inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>divisor</i>  | the divisor       |

**27.2.2.46 static void SDHC\_HAL\_SetClockFrequency ( uint32\_t *baseAddr*, uint32\_t *frequency* ) [inline], [static]**

Parameters

|                  |                        |
|------------------|------------------------|
| <i>baseAddr</i>  | SDHC base address      |
| <i>frequency</i> | the frequency selector |

**27.2.2.47 static void SDHC\_HAL\_SetDataTimeout ( uint32\_t *baseAddr*, uint32\_t *timeout* )  
[inline], [static]**

Parameters

|                 |                            |
|-----------------|----------------------------|
| <i>baseAddr</i> | SDHC base address          |
| <i>timeout</i>  | Data timeout counter value |

**27.2.2.48 static uint32\_t SDHC\_HAL\_GetIntFlags ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

current interrupt flags

**27.2.2.49 static void SDHC\_HAL\_ClearIntFlags ( uint32\_t *baseAddr*, uint32\_t *mask* ) [inline], [static]**

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>baseAddr</i> | SDHC base address                          |
| <i>mask</i>     | to specify interrupts' flags to be cleared |

**27.2.2.50 static uint32\_t SDHC\_HAL\_GetIntSignal ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

currently enabled interrupt signal

**27.2.2.51 static uint32\_t SDHC\_HAL\_GetIntState ( uint32\_t *baseAddr* ) [inline], [static]**

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

currently enabled interrupts' state

**27.2.2.52 static uint32\_t SDHC\_HAL\_GetAc12Error ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

auto cmd12 error status

**27.2.2.53 static uint32\_t SDHC\_HAL\_GetMaxBlockLength ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

the maximum block length support

**27.2.2.54 static uint32\_t SDHC\_HAL\_DoesHostSupportAdma ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if ADMA is supported

27.2.2.55 **static uint32\_t SDHC\_HAL\_DoesHostSupportHighspeed ( uint32\_t *baseAddr* )**  
[**inline**], [**static**]

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if high speed is supported

**27.2.2.56 static uint32\_t SDHC\_HAL\_DoesHostSupportDma ( uint32\_t *baseAddr* )  
[inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if high speed is supported

**27.2.2.57 static uint32\_t SDHC\_HAL\_DoesHostSupportSuspendResume ( uint32\_t  
                  *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if suspend and resume is supported

**27.2.2.58 static uint32\_t SDHC\_HAL\_DoesHostSupportV330 ( uint32\_t *baseAddr* )  
[inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if voltage 3.3 is supported

27.2.2.59 **static uint32\_t SDHC\_HAL\_DoesHostSupportV300 ( uint32\_t *baseAddr* )**  
[**inline**], [**static**]

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if voltage 3.0 is supported

**27.2.2.60 static uint32\_t SDHC\_HAL\_DoesHostSupportV180 ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if voltage 1.8 is supported

**27.2.2.61 static void SDHC\_HAL\_SetWriteWatermarkLevel ( uint32\_t *baseAddr*, uint32\_t *watermark* ) [inline], [static]**

Parameters

|                  |                   |
|------------------|-------------------|
| <i>baseAddr</i>  | SDHC base address |
| <i>watermark</i> | for writing       |

**27.2.2.62 static void SDHC\_HAL\_SetReadWatermarkLevel ( uint32\_t *baseAddr*, uint32\_t *watermark* ) [inline], [static]**

Parameters

|                  |                   |
|------------------|-------------------|
| <i>baseAddr</i>  | SDHC base address |
| <i>watermark</i> | for reading       |

**27.2.2.63 static void SDHC\_HAL\_SetForceEventFlags ( uint32\_t *baseAddr*, uint32\_t *mask* ) [inline], [static]**

Parameters

|                 |                                              |
|-----------------|----------------------------------------------|
| <i>baseAddr</i> | SDHC base address                            |
| <i>mask</i>     | to specify the force events' flags to be set |

**27.2.2.64 static uint32\_t SDHC\_HAL\_IsAdmaLengthMismatchError ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if ADMA error is length mismatch

**27.2.2.65 static bool SDHC\_HAL\_IsSdClockOff ( uint32\_t *baseAddr* ) [inline], [static]**

Checks whether the clock to the SD is enabled.

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

true if enabled

**27.2.2.66 static uint32\_t SDHC\_HAL\_GetAdmaErrorState ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

error state

**27.2.2.67 static uint32\_t SDHC\_HAL\_IsAdmaDescriptionError ( uint32\_t *baseAddr* ) [inline], [static]**

## SDHC HAL

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

if ADMA error is descriptor error

**27.2.2.68 static void SDHC\_HAL\_SetAdmaAddress ( uint32\_t *baseAddr*, uint32\_t *address* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>address</i>  | for ADMA transfer |

**27.2.2.69 static void SDHC\_HAL\_SetExternalDmaRequest ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>enable</i>   | to external DMA   |

**27.2.2.70 static void SDHC\_HAL\_SetExactBlockNumber ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>baseAddr</i> | SDHC base address                                      |
| <i>enable</i>   | to enable exact block number block read for SDIO CMD53 |

**27.2.2.71 static void SDHC\_HAL\_SetBootAckTimeout ( uint32\_t *baseAddr*, uint32\_t *timeout* ) [inline], [static]**

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | SDHC base address               |
| <i>timeout</i>  | boot ack time out counter value |

**27.2.2.72 static void SDHC\_HAL\_SetBootAck ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>baseAddr</i> | SDHC base address       |
| <i>enable</i>   | to enable boot ack mode |

**27.2.2.73 static void SDHC\_HAL\_SetBootMode ( uint32\_t *baseAddr*, sdhc\_hal\_mmcboot\_t *mode* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>mode</i>     | the boot mode     |

**27.2.2.74 static void SDHC\_HAL\_SetFastboot ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>baseAddr</i> | SDHC base address   |
| <i>enable</i>   | to enable fast boot |

**27.2.2.75 static void SDHC\_HAL\_SetAutoStopAtBlockGap ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

## SDHC HAL

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>baseAddr</i> | SDHC base address                                     |
| <i>enable</i>   | to enable auto stop at block gap function, when boot. |

**27.2.2.76 static void SDHC\_HAL\_SetBootBlockCount ( *uint32\_t baseAddr, uint32\_t blockCount* ) [inline], [static]**

Parameters

|                   |                          |
|-------------------|--------------------------|
| <i>baseAddr</i>   | SDHC base address        |
| <i>blockCount</i> | the block count for boot |

**27.2.2.77 static uint32\_t SDHC\_HAL\_GetSpecificationVersion ( *uint32\_t baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

specification version

**27.2.2.78 static uint32\_t SDHC\_HAL\_GetVendorVersion ( *uint32\_t baseAddr* ) [inline], [static]**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

vendor version

**27.2.2.79 uint32\_t SDHC\_HAL.GetResponse ( *uint32\_t baseAddr, uint32\_t index* )**

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | SDHC base address                       |
| <i>index</i>    | of response register, range from 0 to 3 |

### **27.2.2.80 void SDHC\_HAL\_SetIntSignal ( uint32\_t *baseAddr*, bool *enable*, uint32\_t *mask* )**

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | SDHC base address                       |
| <i>enable</i>   | enable or disable                       |
| <i>mask</i>     | to specify interrupts to be isEnableddd |

### **27.2.2.81 void SDHC\_HAL\_SetIntState ( uint32\_t *baseAddr*, bool *enable*, uint32\_t *mask* )**

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>baseAddr</i> | SDHC base address                          |
| <i>enable</i>   | enable or disable                          |
| <i>mask</i>     | to specify interrupts' state to be enabled |

### **27.2.2.82 uint32\_t SDHC\_HAL\_Reset ( uint32\_t *baseAddr*, uint32\_t *type*, uint32\_t *timeout* )**

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
| <i>type</i>     | the type of reset |
| <i>timeout</i>  | timeout for reset |

Returns

0 on success, else on error

### **27.2.2.83 uint32\_t SDHC\_HAL\_InitCard ( uint32\_t *baseAddr*, uint32\_t *timeout* )**

## SDHC HAL

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>baseAddr</i> | SDHC base address           |
| <i>timeout</i>  | timeout for initialize card |

Returns

0 on success, else on error

### 27.2.2.84 IRQn\_Type SDHC\_HAL\_GetIrqId ( uint32\_t *baseAddr* )

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

Returns

IRQ number for specific SDHC instance

### 27.2.2.85 void SDHC\_HAL\_Init ( uint32\_t *baseAddr* )

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | SDHC base address |
|-----------------|-------------------|

## 27.3 SDHC Peripheral Driver

### 27.3.1 Overview

This section describes the programming interface of the SDCH Peripheral driver. The SDHC driver configures the secure digital host controller and provides an easy way to operate the SDHC module.

### 27.3.2 SDHC Initialization

To initialize the SDHC module, call the `SDHC_DRV_Init()` function and pass in the configuration data structure.

This is an example code to initialize and configure the driver:

```
// Define device configuration.
sdhc_user_config_t config = {0};

config.clock = 0;

// Initialize
if (SDHC_DRV_Init(instance, &host, &config) != kStatus_SDHC_NoError)
{
 // error occurs //
}
else
{
 // SDHC has been successfully initialized //
}
```

### 27.3.3 SDHC Issuing a request to the card

The SDHC driver provides a simple way to send commands to and retrieve response/data from the card.

This is a example to send the SD\_SWITCH command to the card.

```
sdhc_request_t req = {0};
sdhc_data_t data = {0};

req.cmdIndex = kSdSwitch; // Set command index //
req.argument = mode << 31 | 0x00FFFFFF; // Set argument //
req.argument &= ~((0xF) << (group * 4)); // Set command flags //
req.flags = FSL_SDHC_REQ_FLAGS_DATA_READ; // Set response type //
req.respType = kSdhcRespTypeR1; // Set data block size //
 // Set data block count //
 // Set data buffer //

data.blockSize = 64; // Set data block size //
data.blockCount = 1; // Set data block count //
data.buffer = response; // Set data buffer //

// link data, command with request //
data.req = &req;
req.data = &data;
if (kStatus_SDHC_NoError != SDHC_DRV_IssueRequestBlocking(
 card->hostInstance,
 &req,
 FSL_SDCARD_REQUEST_TIMEOUT);) // issue request //
```

## SDHC Peripheral Driver

```
{
 // error occurs //
}
else
{
 // request has been issued successfully //
}
```

This section describes the programming interface of the SDCH Peripheral Driver.

### SDHC PD FUNCTION

- `sdhc_status_t SDHC_DRV_Init (uint32_t instance, sdhc_host_t *host, const sdhc_user_config_t *config)`  
*Initializes the Host controller by a specific instance index.*
- `void SDHC_DRV_Shutdown (uint32_t instance)`  
*Destroys the host controller.*
- `sdhc_status_t SDHC_DRV_DetectCard (uint32_t instance)`  
*Checks whether the card is present on a specified host controller.*
- `sdhc_status_t SDHC_DRV_ConfigClock (uint32_t instance, uint32_t clock)`  
*Sets the clock frequency of the host controller.*
- `sdhc_status_t SDHC_DRV_SetBusWidth (uint32_t instance, sdhc_buswidth_t busWidth)`  
*Sets the bus width of the host controller.*
- `sdhc_status_t SDHC_DRV_IssueRequestBlocking (uint32_t instance, sdhc_request_t *req, uint32_t timeoutInMs)`  
*Issues the request on a specific host controller and returns on completion.*
- `void SDHC_DRV_DoIrq (uint32_t instance)`  
*IRQ handler for SDHC.*

#### 27.3.4 Function Documentation

##### 27.3.4.1 `sdhc_status_t SDHC_DRV_Init ( uint32_t instance, sdhc_host_t * host, const sdhc_user_config_t * config )`

This function initializes the SDHC module according to the given initialization configuration structure including the clock frequency, bus width, and card detect callback.

Parameters

|                       |                                                                   |
|-----------------------|-------------------------------------------------------------------|
| <code>instance</code> | the specific instance index                                       |
| <code>host</code>     | pointer to a place storing the <code>sdhc_host_t</code> structure |

|               |                                   |
|---------------|-----------------------------------|
| <i>config</i> | initialization configuration data |
|---------------|-----------------------------------|

Returns

kStatus\_SDHC\_NoError if success

#### 27.3.4.2 void SDHC\_DRV\_Shutdown ( uint32\_t *instance* )

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | the instance index of host controller |
|-----------------|---------------------------------------|

#### 27.3.4.3 sdhc\_status\_t SDHC\_DRV\_DetectCard ( uint32\_t *instance* )

This function checks if there's a card inserted in the SDHC.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | the instance index of host controller |
|-----------------|---------------------------------------|

Returns

kStatus\_SDHC\_NoError on success

#### 27.3.4.4 sdhc\_status\_t SDHC\_DRV\_ConfigClock ( uint32\_t *instance*, uint32\_t *clock* )

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>instance</i> | the instance index of host controller         |
| <i>clock</i>    | the desired frequency to be set to controller |

Returns

kStatus\_SDHC\_NoError on success

#### 27.3.4.5 sdhc\_status\_t SDHC\_DRV\_SetBusWidth ( uint32\_t *instance*, sdhc\_buswidth\_t *busWidth* )

## SDHC Peripheral Driver

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>instance</i> | the instance index of host controller         |
| <i>busWidth</i> | the desired bus width to be set to controller |

Returns

kStatus\_SDHC\_NoError on success

### 27.3.4.6 `sdhc_status_t SDHC_DRV_IssueRequestBlocking ( uint32_t instance, sdhc_request_t * req, uint32_t timeoutInMs )`

This function issues the request to the card on a specific SDHC. The command is sent and is blocked as long as the response/data is sending back from the card.

Parameters

|                    |                                       |
|--------------------|---------------------------------------|
| <i>instance</i>    | the instance index of host controller |
| <i>req</i>         | the pointer to the request            |
| <i>timeoutInMs</i> | timeout value in microseconds         |

Returns

kStatus\_SDHC\_NoError on success

### 27.3.4.7 `void SDHC_DRV_Dolrq ( uint32_t instance )`

This function deals with IRQs on the given host controller.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | the instance index of host controller |
|-----------------|---------------------------------------|

## 27.4 SDHC Data Types

### 27.4.1 Overview

This section describes the SDHC Data Types.

### Data Structures

- struct `sdhc_user_config_t`  
*SDHC Initialization Configuration Structure. [More...](#)*
- struct `sdhc_host_t`  
*SDHC Host Device Structure. [More...](#)*
- struct `sdhc_data_t`  
*SDHC Data Structure. [More...](#)*
- struct `sdhc_request_t`  
*SDHC Request Structure. [More...](#)*

## SDHC Data Types

### Enumerations

- enum `sdhc_status_t` {  
    kStatus\_SDHC\_NoError = 0,  
    kStatus\_SDHC\_InitFailed,  
    kStatus\_SDHC\_SetClockFailed,  
    kStatus\_SDHC\_SetCardToIdle,  
    kStatus\_SDHC\_SetCardBlockSizeFailed,  
    kStatus\_SDHC\_SendAppOpCondFailed,  
    kStatus\_SDHC\_AllSendCidFailed,  
    kStatus\_SDHC\_SendRcaFailed,  
    kStatus\_SDHC\_SendCsdFailed,  
    kStatus\_SDHC\_SendScrFailed,  
    kStatus\_SDHC\_SelectCardFailed,  
    kStatus\_SDHC\_SwitchHighSpeedFailed,  
    kStatus\_SDHC\_SetCardWideBusFailed,  
    kStatus\_SDHC\_SetBusWidthFailed,  
    kStatus\_SDHC\_SendCardStatusFailed,  
    kStatus\_SDHC\_StopTransmissionFailed,  
    kStatus\_SDHC\_CardEraseBlocksFailed,  
    kStatus\_SDHC\_InvalidIORange,  
    kStatus\_SDHC\_BlockSizeNotSupportError,  
    kStatus\_SDHC\_HostIsAlreadyInitiated,  
    kStatus\_SDHC\_HostNotSupport,  
    kStatus\_SDHC\_HostIsBusyError,  
    kStatus\_SDHC\_DataPrepareError,  
    kStatus\_SDHC\_WaitTimeoutError,  
    kStatus\_SDHC\_OutOfMemory,  
    kStatus\_SDHC\_IoError,  
    kStatus\_SDHC\_CmdIoError,  
    kStatus\_SDHC\_DataIoError,  
    kStatus\_SDHC\_InvalidParameter,  
    kStatus\_SDHC\_RequestFailed,  
    kStatus\_SDHC\_RequestCardStatusError,  
    kStatus\_SDHC\_SwitchFailed,  
    kStatus\_SDHC\_NotSupportYet,  
    kStatus\_SDHC\_TimeoutError,  
    kStatus\_SDHC\_CardNotSupport,  
    kStatus\_SDHC\_CmdError,  
    kStatus\_SDHC\_DataError,  
    kStatus\_SDHC\_DmaAddressError,  
    kStatus\_SDHC\_Failed,  
    kStatus\_SDHC\_NoMedium,  
    kStatus\_SDHC\_UnknownStatus }
- enum `sdhc_cd_type_t` {

```

kSdhcCardDetectGpio = 1,
kSdhcCardDetectDat3,
kSdhcCardDetectCdPin,
kSdhcCardDetectPollDat3,
kSdhcCardDetectPollCd }
• enum sdhc_power_mode_t {
 kSdhcPowerModeRunning = 0,
 kSdhcPowerModeSuspended,
 kSdhcPowerModeStopped }
• enum sdhc_buswidth_t {
 kSdhcBusWidth1Bit = 1,
 kSdhcBusWidth4Bit,
 kSdhcBusWidth8Bit }
• enum sdhc_transfer_mode_t {
 kSdhcTransModePio = 1,
 kSdhcTransModeSdma,
 kSdhcTransModeAdma1,
 kSdhcTransModeAdma2 }
• enum sdhc_resp_type_t {
 kSdhcRespTypeNone = 0,
 kSdhcRespTypeR1,
 kSdhcRespTypeR1b,
 kSdhcRespTypeR2,
 kSdhcRespTypeR3,
 kSdhcRespTypeR4,
 kSdhcRespTypeR5,
 kSdhcRespTypeR5b,
 kSdhcRespTypeR6,
 kSdhcRespTypeR7 }

```

## 27.4.2 Data Structure Documentation

### 27.4.2.1 struct sdhc\_user\_config\_t

Defines the configuration data structure to initialize the SDHC.

#### Data Fields

- uint32\_t clock  
*Clock rate.*
- sdhc\_transfer\_mode\_t transMode  
*SDHC transfer mode.*
- sdhc\_cd\_type\_t cdType  
*Card detection type.*
- void(\* cardDetectCallback )(bool inserted)

## SDHC Data Types

- *Callback function for card detect occurs.*  
• void(\* **cardIntCallback** )(void)
- *Callback function for card interrupt occurs.*  
• void(\* **blockGapCallback** )(void)
- *Callback function for block gap occurs.*

### 27.4.2.2 struct sdhc\_host\_t

Defines the Host device structure which includes both the static and the runtime SDHC information.

#### Data Fields

- uint32\_t **instance**  
*Host instance index.*
- **sdhc\_cd\_type\_t cdType**  
*Host controller card detection type.*
- **sdhc\_hal\_endian\_t endian**  
*Endian mode the host's working at.*
- uint32\_t **swFeature**  
*Host controller driver features.*
- uint32\_t **flags**  
*Host flags.*
- uint32\_t **busWidth**  
*Current busWidth.*
- uint32\_t **caps**  
*Host capability.*
- uint32\_t **ocrSupported**  
*Supported OCR.*
- uint32\_t **clock**  
*Current clock frequency.*
- **sdhc\_power\_mode\_t powerMode**  
*Current power mode.*
- uint32\_t **maxClock**  
*Maximum clock supported.*
- uint32\_t **maxBlockSize**  
*Maximum block size supported.*
- uint32\_t **maxBlockCount**  
*Maximum block count supported.*
- uint32\_t \* **admaTableAddress**  
*ADMA table address.*
- struct SdhcRequest \* **currentReq**  
*Associated request.*
- void(\* **cardIntCallback** )(void)  
*Callback function for card interrupt occurs.*
- void(\* **cardDetectCallback** )(bool inserted)  
*Callback function for card detect occurs.*
- void(\* **blockGapCallback** )(void)  
*Callback function for block gap occurs.*

### 27.4.2.3 struct sdhc\_data\_t

Defines the SDHC data structure including the block size/count and flags.

#### Data Fields

- struct SdhcRequest \* **req**  
*Associated request.*
- uint32\_t **blockSize**  
*Block size.*
- uint32\_t **blockCount**  
*Block count.*
- uint32\_t **bytesTransferred**  
*Transferred buffer.*
- uint32\_t \* **buffer**  
*Data buffer.*

### 27.4.2.4 struct sdhc\_request\_t

Defines the SDHC request structure including the command index, argument, flags, response, and data.

#### Data Fields

- uint32\_t **cmdIndex**  
*Command index.*
- uint32\_t **argument**  
*Command argument.*
- uint32\_t **flags**  
*Flags.*
- **sdhc\_resp\_type\_t respType**  
*Response type.*
- volatile uint32\_t **error**  
*Command error code.*
- uint32\_t **cardErrStatus**  
*Card error status from response 1.*
- uint32\_t **response** [4]  
*Response for this command.*
- semaphore\_t \* **complete**  
*Request completion sync object.*
- struct SdhcData \* **data**  
*Data associated with request.*

## SDHC Data Types

### 27.4.3 Enumeration Type Documentation

#### 27.4.3.1 enum sdhc\_status\_t

Enumerator

*kStatus\_SDHC\_NoError* No error.  
*kStatus\_SDHC\_InitFailed* Driver initialization failed.  
*kStatus\_SDHC\_SetClockFailed* Failed to set clock of host controller.  
*kStatus\_SDHC\_SetCardToIdle* Failed to set card to idle.  
*kStatus\_SDHC\_SetCardBlockSizeFailed* Failed to set card block size.  
*kStatus\_SDHC\_SendAppOpCondFailed* Failed to send app\_op\_cond command.  
*kStatus\_SDHC\_AllSendCidFailed* Failed to send all\_send\_cid command.  
*kStatus\_SDHC\_SendRcaFailed* Failed to send send\_rca command.  
*kStatus\_SDHC\_SendCsdFailed* Failed to send send\_csd command.  
*kStatus\_SDHC\_SendScrFailed* Failed to send send\_scr command.  
*kStatus\_SDHC\_SelectCardFailed* Failed to send select\_card command.  
*kStatus\_SDHC\_SwitchHighSpeedFailed* Failed to switch to high speed mode.  
*kStatus\_SDHC\_SetCardWideBusFailed* Failed to set card's bus mode.  
*kStatus\_SDHC\_SetBusWidthFailed* Failed to set host's bus mode.  
*kStatus\_SDHC\_SendCardStatusFailed* Failed to send card status.  
*kStatus\_SDHC\_StopTransmissionFailed* Failed to stop transmission.  
*kStatus\_SDHC\_CardEraseBlocksFailed* Failed to erase blocks.  
*kStatus\_SDHC\_InvalidIORange* Invalid read/write/erase address range.  
*kStatus\_SDHC\_BlockSizeNotSupportError* Unsupported block size.  
*kStatus\_SDHC\_HostIsAlreadyInitiated* Host controller is already initialized.  
*kStatus\_SDHC\_HostNotSupport* Host not error.  
*kStatus\_SDHC\_HostIsBusyError* Bus busy error.  
*kStatus\_SDHC\_DataPrepareError* Data preparation error.  
*kStatus\_SDHC\_WaitTimeoutError* Wait timeout error.  
*kStatus\_SDHC\_OutOfMemory* Out of memory error.  
*kStatus\_SDHC\_IoError* General IO error.  
*kStatus\_SDHC\_CmdIoError* CMD I/O error.  
*kStatus\_SDHC\_DataIoError* Data I/O error.  
*kStatus\_SDHC\_InvalidParameter* Invalid parameter error.  
*kStatus\_SDHC\_RequestFailed* Request failed.  
*kStatus\_SDHC\_RequestCardStatusError* Status error.  
*kStatus\_SDHC\_SwitchFailed* Switch failed.  
*kStatus\_SDHC\_NotSupportYet* Not support.  
*kStatus\_SDHC\_TimeoutError* Timeout error.  
*kStatus\_SDHC\_CardNotSupport* Card does not support.  
*kStatus\_SDHC\_CmdError* CMD error.  
*kStatus\_SDHC\_DataError* Data error.  
*kStatus\_SDHC\_DmaAddressError* DMA address error.  
*kStatus\_SDHC\_Failed* General failed.

*kStatus\_SDHC\_NoMedium* No medium error.

*kStatus\_SDHC\_UnknownStatus* Unknown if card is present.

#### 27.4.3.2 enum sdhc\_cd\_type\_t

Enumerator

*kSdhcCardDetectGpio* Use GPIO for card detection.

*kSdhcCardDetectDat3* Use DAT3 for card detection.

*kSdhcCardDetectCdPin* Use host controller dedicate CD pin for card detection.

*kSdhcCardDetectPollDat3* Poll DAT3 for card detection.

*kSdhcCardDetectPollCd* Poll host controller dedicate CD pin for card detection.

#### 27.4.3.3 enum sdhc\_power\_mode\_t

Enumerator

*kSdhcPowerModeRunning* SDHC is running.

*kSdhcPowerModeSuspended* SDHC is suspended.

*kSdhcPowerModeStopped* SDHC is stopped.

#### 27.4.3.4 enum sdhc\_buswidth\_t

Enumerator

*kSdhcBusWidth1Bit* 1-bit bus width.

*kSdhcBusWidth4Bit* 4-bit bus width.

*kSdhcBusWidth8Bit* 8-bit bus width.

#### 27.4.3.5 enum sdhc\_transfer\_mode\_t

Enumerator

*kSdhcTransModePio* Transfer mode: PIO.

*kSdhcTransModeSdma* Transfer mode: SDMA.

*kSdhcTransModeAdma1* Transfer mode: ADMA1.

*kSdhcTransModeAdma2* Transfer mode: ADMA2.

## SDHC Data Types

### 27.4.3.6 enum sdhc\_resp\_type\_t

Enumerator

*kSdhcRespTypeNone* Response type: none.

*kSdhcRespTypeR1* Response type: R1.

*kSdhcRespTypeR1b* Response type: R1b.

*kSdhcRespTypeR2* Response type: R2.

*kSdhcRespTypeR3* Response type: R3.

*kSdhcRespTypeR4* Response type: R4.

*kSdhcRespTypeR5* Response type: R5.

*kSdhcRespTypeR5b* Response type: R5b.

*kSdhcRespTypeR6* Response type: R6.

*kSdhcRespTypeR7* Response type: R7.

## 27.5 SDHC Standard Definition

### 27.5.1 Overview

This section describes the host controller standard definition.

#### Macros

- #define **SDHC\_DMA\_ADDRESS** (0x00U)  
*SDHC DMA ADDRESS REG.*
- #define **SDHC\_BLOCK\_SIZE** (0x04U)  
*SDHC BLOCK SIZE REG.*
- #define **SDHC\_BLOCK\_COUNT** (0x06U)  
*SDHC BLOCK COUNT REG.*
- #define **SDHC\_ARGUMENT** (0x08U)  
*SDHC ARGUMENT REG.*
- #define **SDHC\_TRANSFER\_MODE** (0x0C)  
*SDHC TRANSFER MODE REG.*
- #define **SDHC\_TRNSM\_DMA\_EN** (0x01U)  
*SDHC TRANSFER MODE DMA ENABLE BIT.*
- #define **SDHC\_TRNSM\_BLKCNT\_EN** (0x02U)  
*SDHC TRANSFER MODE BLOCK COUNT ENABLE BIT.*
- #define **SDHC\_TRNSM\_AUTOCMD12** (0x04U)  
*SDHC TRANSFER MODE AUTO CMD12 BIT.*
- #define **SDHC\_TRNSM\_AUTOCMD23** (0x08U)  
*SDHC TRANSFER MODE AUTO CMD23 BIT.*
- #define **SDHC\_TRNSM\_READ** (0x10U)  
*SDHC TRANSFER MODE READ DATA BIT.*
- #define **SDHC\_TRNSM\_MULTI** (0x20U)  
*SDHC TRANSFER MODE MULTIBLOCK BIT.*
- #define **SDHC\_COMMAND** (0x0E)  
*SDHC COMMAND REG.*
- #define **SDHC\_CMD\_RESPTYPE\_LSF** (0U)  
*SDHC COMMAND RESPONSE TYPE SHIFT.*
- #define **SDHC\_CMD\_RESPTYPE\_MASK** (0x03U)  
*SDHC COMMAND RESPONSE MASK.*
- #define **SDHC\_CMD\_CRC\_CHK** (0x08U)  
*SDHC COMMAND CRC CHECKING BIT.*
- #define **SDHC\_CMD\_INDEX\_CHK** (0x10U)  
*SDHC COMMAND INDEX CHECKING BIT.*
- #define **SDHC\_CMD\_DATA\_PRSNT** (0x20U)  
*SDHC COMMAND DATA PRESENT BIT.*
- #define **SDHC\_CMD\_CMDTYPE\_LSF** (6U)  
*SDHC COMMAND COMMAND TYPE SHIFT.*
- #define **SDHC\_CMD\_CMDTYPE\_MASK** (0xC0U)  
*SDHC COMMAND COMMAND TYPE MASK.*
- #define **SDHC\_CMD\_CMDINDEX\_LSF** (8U)  
*SDHC COMMAND COMMAND INDEX SHIFT.*
- #define **SDHC\_CMD\_CMDINDEX\_MASK** (0x3F)  
*SDHC COMMAND COMMAND INDEX MASK.*

## SDHC Standard Definition

- #define **SDHC\_RESPONSE** (0x10U)  
*SDHC RESPONSE REG.*
- #define **SDHC\_BUFFER** (0x20U)  
*SDHC BUFFER REG.*
- #define **SDHC\_PRESENT\_STATE** (0x24U)  
*SDHC PRESENT STATE REG.*
- #define **SDHC\_PRST\_CMD\_INHIBIT** (0x1U)  
*SDHC PRESENT STATE CMD INHIBIT BIT.*
- #define **SDHC\_PRST\_DATA\_INHIBIT** (0x1 << 1)  
*SDHC PRESENT STATE DATA INHIBIT BIT.*
- #define **SDHC\_PRST\_DLA** (0x1 << 2)  
*SDHC PRESENT STATE DATA LINE ACTIVE BIT.*
- #define **SDHC\_PRST\_RETUNE\_REQ** (0x1 << 3)  
*SDHC PRESENT STATE RETUNE REQUEST BIT.*
- #define **SDHC\_PRST\_WR\_TRANS\_A** (0x1 << 8)  
*SDHC PRESENT STATE WRITE TRANSFER ACTIVE BIT.*
- #define **SDHC\_PRST\_RD\_TRANS\_A** (0x1 << 9)  
*SDHC PRESENT STATE READ TRANSFER ACTIVE BIT.*
- #define **SDHC\_PRST\_BUFF\_WR** (0x1 << 10)  
*SDHC PRESENT STATE BUFFER WRITE ENABLE BIT.*
- #define **SDHC\_PRST\_BUFF\_RD** (0x1 << 11)  
*SDHC PRESENT STATE BUFFER READ ENABLE BIT.*
- #define **SDHC\_PRST\_CARD\_INSERTED** (0x1 << 16)  
*SDHC PRESENT STATE CARD INSERTED BIT.*
- #define **SDHC\_PRST\_CSS** (0x1 << 17)  
*SDHC PRESENT STATE CARD STATE STABLE BIT.*
- #define **SDHC\_PRST\_CD\_LVL** (0x1 << 18)  
*SDHC PRESENT STATE CARD DETECT PIN LEVEL BIT.*
- #define **SDHC\_PRST\_WP\_LVL** (0x1 << 19)  
*SDHC PRESENT STATE WRITE PROTECT PIN LEVEL BIT.*
- #define **SDHC\_PRST\_DLSL\_0\_3\_LSF** (20U)  
*SDHC PRESENT STATE DAT[3:0] LINE LEVEL SHIFT.*
- #define **SDHC\_PRST\_DLSL\_0\_3\_MASK** (0x0F000000U)  
*SDHC PRESENT STATE DAT[3:0] LINE LEVEL MASK.*
- #define **SDHC\_PRST\_CMD\_LVL** (0x1 << 24)  
*SDHC PRESENT STATE CMD LINE LEVEL BIT.*
- #define **SDHC\_HOST\_CONTROL1** (0x28U)  
*SDHC HOST CONTROL1 REG.*
- #define **SDHC\_CTRL\_LED** (0x01U)  
*SDHC HOST CONTROL1 LED CONTROL BIT.*
- #define **SDHC\_CTRL\_4BIT** (0x02U)  
*SDHC HOST CONTROL1 DATA TRANSFER WIDTH BIT.*
- #define **SDHC\_CTRL\_HISPD** (0x04U)  
*SDHC HOST CONTROL1 HIGH SPEED ENABLE BIT.*
- #define **SDHC\_CTRL\_DMA\_LSF** (0x3U)  
*SDHC HOST CONTROL1 DMA SELECT SHIFT.*
- #define **SDHC\_CTRL\_DMA\_MASK** (0x18U)  
*SDHC HOST CONTROL1 DMA SELECT MASK.*
- #define **SDHC\_CTRL\_DMA\_SDMA** (0x0U)  
*SDHC HOST CONTROL1 DMA SELECT SDMA.*
- #define **SDHC\_CTRL\_DMA\_ADMA32** (0x2U)

- *SDHC HOST CONTROL1 DMA SELECT ADMA32.*
- #define **SDHC\_CTRL\_DMA\_ADMA64** (0x3U)
  - SDHC HOST CONTROL1 DMA SELECT ADMA64.*
- #define **SDHC\_CTRL\_8BIT** (0x20U)
  - SDHC HOST CONTROL1 EXTENDED DATA TRANSFER WIDTH BIT.*
- #define **SDHC\_CTRL\_CD\_TEST\_LVL** (0x40U)
  - SDHC HOST CONTROL1 CARD DETECT TEST LEVEL BIT.*
- #define **SDHC\_CTRL\_CD\_SSELECT** (0x80U)
  - SDHC HOST CONTROL1 CARD DETECT SIGNAL SELECTION BIT.*
- #define **SDHC\_POWER\_CONTROL** (0x29U)
  - SDHC POWER CONTROL REG.*
  - #define **SDHC\_POWER\_ON** (0x01U)
    - SDHC POWER CONTROL SD BUS POWER.*
  - #define **SDHC\_POWER\_180** (0x0A)
    - SDHC POWER CONTROL SD BUS POWER 1.8V.*
  - #define **SDHC\_POWER\_300** (0x0C)
    - SDHC POWER CONTROL SD BUS POWER 3.0V.*
  - #define **SDHC\_POWER\_330** (0x0E)
    - SDHC POWER CONTROL SD BUS POWER 3.3V.*
  - #define **SDHC\_BLOCK\_GAP\_CTRL** (0x2A)
    - SDHC BLOCK GAP CONTROL REG.*
    - #define **SDHC\_BGCTRL\_STPATGAPREQ** (0x01U)
      - SDHC BLOCK GAP CONTROL STOP AT BLOCK GAP BIT.*
    - #define **SDHC\_BGCTRL\_CNTNREQ** (0x02U)
      - SDHC BLOCK GAP CONTROL CONTINUE REQUEST BIT.*
    - #define **SDHC\_BGCTRL\_READWAIT** (0x04U)
      - SDHC BLOCK GAP CONTROL READ WAIT CONTROL BIT.*
    - #define **SDHC\_BGCTRL\_INTRATGAP** (0x08U)
      - SDHC BLOCK GAP CONTROL INTERRUPT AT BLOCK GAP BIT.*
    - #define **SDHC\_WAKEUP\_CONTROL** (0x2B)
      - SDHC WAKEUP CONTROL REG.*
      - #define **SDHC\_WAKE\_ON\_INT** (0x01U)
        - SDHC WAKEUP CONTROL WAKEUP ON CARD INTERRUPT BIT.*
      - #define **SDHC\_WAKE\_ON\_INSERT** (0x02U)
        - SDHC WAKEUP CONTROL WAKEUP ON CARD INSERTION BIT.*
      - #define **SDHC\_WAKE\_ON\_REMOVE** (0x04U)
        - SDHC WAKEUP CONTROL WAKEUP ON CARD REMOVAL BIT.*
      - #define **SDHC\_CLOCK\_CONTROL** (0x2C)
        - SDHC CLOCK CONTROL REG.*
        - #define **SDHC\_CLK\_INTCLK\_EN** (0x0001U)
          - SDHC CLOCK CONTROL INTERNAL CLOCK ENABLE BIT.*
        - #define **SDHC\_CLK\_INTCLK\_STB** (0x0002U)
          - SDHC CLOCK CONTROL INTERNAL CLOCK STABLE BIT.*
        - #define **SDHC\_CLK\_SDCLK\_EN** (0x0004U)
          - SDHC CLOCK CONTROL SD CLOCK ENABLE BIT.*
        - #define **SDHC\_CLK\_CLKGEN\_PRG\_SEL** (0x0020U)
          - SDHC CLOCK CONTROL CLOCK GENERATOR SELECTOR BIT.*
        - #define **SDHC\_CLK\_FREQ\_U\_LSF** (6U)
          - SDHC CLOCK CONTROL UPPER BITS OF FREQUENCY SELECTOR SHIFT.*
        - #define **SDHC\_CLK\_FREQ\_U\_MASK** (0x00C0U)
          - SDHC CLOCK CONTROL UPPER BITS OF FREQUENCY SELECTOR MASK.*

## SDHC Standard Definition

- #define **SDHC\_CLK\_FREQ\_SEL\_LSF** (8U)  
*SDHC CLOCK CONTROL FREQUENCY SELECTOR SHIFT.*
- #define **SDHC\_CLK\_FREQ\_SEL\_MASK** (0xFF00U)  
*SDHC CLOCK CONTROL FREQUENCY SELECTOR MASK.*
- #define **SDHC\_TIMEOUT\_CONTROL** (0x2E)  
*SDHC TIMEOUT CONTROL REG.*
- #define **SDHC\_SOFTWARE\_RESET** (0x2F)  
*SDHC SOFTWARE RESET REG.*
- #define **SDHC\_RESET\_ALL** (0x01U)  
*SDHC SOFTWARE RESET RESET FOR ALL.*
- #define **SDHC\_RESET\_CMD** (0x02U)  
*SDHC SOFTWARE RESET RESET FOR CMD LINE.*
- #define **SDHC\_RESET\_DATA** (0x04U)  
*SDHC SOFTWARE RESET RESET FOR DATA LINE.*
- #define **SDHC\_INT\_STATUS** (0x30U)  
*SDHC NORMAL INTERRUPT STATUS REG.*
- #define **SDHC\_INT\_ENABLE** (0x34U)  
*SDHC NORMAL INTERRUPT STATUS ENABLE REG.*
- #define **SDHC\_SIGNAL\_ENABLE** (0x38U)  
*SDHC NORMAL INTERRUPT SIGNAL REG.*
- #define **SDHC\_INT\_CMD\_DONE** (0x1U << 0)  
*SDHC NORMAL INTERRUPT CMD COMPLETE EVENT BIT.*
- #define **SDHC\_INT\_TRANSFER\_DONE** (0x1U << 1)  
*SDHC NORMAL INTERRUPT TRANSFER COMPLETE EVENT BIT.*
- #define **SDHC\_INT\_BLKGAP\_EVENT** (0x1U << 2)  
*SDHC NORMAL INTERRUPT BLOCK GAP EVENT BIT.*
- #define **SDHC\_INT\_DMA** (0x1U << 3)  
*SDHC NORMAL INTERRUPT DMA EVENT BIT.*
- #define **SDHC\_INT\_WBUF\_READY** (0x1U << 4)  
*SDHC NORMAL INTERRUPT WRITE BUFFER READY EVENT BIT.*
- #define **SDHC\_INT\_RBUF\_READY** (0x1U << 5)  
*SDHC NORMAL INTERRUPT READ BUFFER READY EVENT BIT.*
- #define **SDHC\_INT\_CARD\_INSERT** (0x1U << 6)  
*SDHC NORMAL INTERRUPT CARD INSERTION EVENT BIT.*
- #define **SDHC\_INT\_CARD\_REMOVE** (0x1U << 7)  
*SDHC NORMAL INTERRUPT CARD REMOVAL EVENT BIT.*
- #define **SDHC\_INT\_CARD\_INTR** (0x1U << 8)  
*SDHC NORMAL INTERRUPT CARD INTERRUPT BIT.*
- #define **SDHC\_INT\_INT\_A** (0x1U << 9)  
*SDHC NORMAL INTERRUPT INT\_A EVENT BIT.*
- #define **SDHC\_INT\_INT\_B** (0x1U << 10)  
*SDHC NORMAL INTERRUPT INT\_B EVENT BIT.*
- #define **SDHC\_INT\_INT\_C** (0x1U << 11)  
*SDHC NORMAL INTERRUPT INT\_C EVENT BIT.*
- #define **SDHC\_INT\_RETUNING** (0x1U << 12)  
*SDHC NORMAL INTERRUPT RETUNING EVENT BIT.*
- #define **SDHC\_INT\_ERROR\_INTR** (0x1U << 15)  
*SDHC NORMAL INTERRUPT ERROR INTERRUPT BIT.*
- #define **SDHC\_INT\_E\_CMD\_TIMEOUT** (0x1U << 16)  
*SDHC NORMAL INTERRUPT CMD TIMEOUT ERROR BIT.*
- #define **SDHC\_INT\_E\_CMD\_CRC** (0x1U << 17)

- #define **SDHC\_INT\_E\_CMD\_END\_BIT** (0x1U << 18)  
SDHC NORMAL INTERRUPT CMD INDEX ERROR BIT.
- #define **SDHC\_INT\_E\_CMD\_INDEX** (0x1U << 19)  
SDHC NORMAL INTERRUPT CMD END BIT ERROR BIT.
- #define **SDHC\_INT\_E\_DATA\_TIMEOUT** (0x1U << 20)  
SDHC NORMAL INTERRUPT DATA TIMEOUT ERROR BIT.
- #define **SDHC\_INT\_E\_DATA\_CRC** (0x1U << 21)  
SDHC NORMAL INTERRUPT DATA CRC ERROR BIT.
- #define **SDHC\_INT\_E\_DATA\_END\_BIT** (0x1U << 22)  
SDHC NORMAL INTERRUPT DATA END BIT ERROR BIT.
- #define **SDHC\_INT\_E\_CUR\_LIMIT** (0x1U << 23)  
SDHC NORMAL INTERRUPT CURRENT LIMIT ERROR BIT.
- #define **SDHC\_INT\_E\_AUTOCMD12** (0x1U << 24)  
SDHC NORMAL INTERRUPT AUTO CMD12 ERROR BIT.
- #define **SDHC\_INT\_E\_ADMA** (0x1U << 25)  
SDHC NORMAL INTERRUPT ADMA ERROR BIT.
- #define **SDHC\_INT\_E\_TUNING** (0x1U << 26)  
SDHC NORMAL INTERRUPT TUNING ERROR BIT.
- #define **SDHC\_ACMD12\_ERROR** (0x3CU)  
SDHC AUTO CMD12 ERROR REG.
- #define **SDHC\_HOST\_CONTROL2** (0x3EU)  
SDHC HOST CONTROL2 REG.
- #define **SDHC\_CTRL2\_UHS\_MASK** (0x0007U)  
SDHC HOST CONTROL2 UHS MODE MASK.
- #define **SDHC\_CTRL2\_UHS\_SDR12** (0x0000U)  
SDHC HOST CONTROL2 UHS-I SDR12.
- #define **SDHC\_CTRL2\_UHS\_SDR25** (0x0001U)  
SDHC HOST CONTROL2 UHS-I SDR25.
- #define **SDHC\_CTRL2\_UHS\_SDR50** (0x0002U)  
SDHC HOST CONTROL2 UHS-I SDR50.
- #define **SDHC\_CTRL2\_UHS\_SDR104** (0x0003U)  
SDHC HOST CONTROL2 UHS-I SDR104.
- #define **SDHC\_CTRL2\_UHS\_DDR50** (0x0004U)  
SDHC HOST CONTROL2 UHS-I DDR50.
- #define **SDHC\_CTRL2\_HS\_SDR200** (0x0005U)  
SDHC HOST CONTROL2 HS SDR2000.
- #define **SDHC\_CTRL2\_VDD\_180** (0x0008U)  
SDHC HOST CONTROL2 1.8V SINGALING ENABLE.
- #define **SDHC\_CTRL2\_DRV\_TYPE\_MASK** (0x0030U)  
SDHC HOST CONTROL2 DRIVE MASK.
- #define **SDHC\_CTRL2\_DRV\_TYPE\_B** (0x0000U)  
SDHC HOST CONTROL2 DRIVE TYPE B.
- #define **SDHC\_CTRL2\_DRV\_TYPE\_A** (0x0010U)  
SDHC HOST CONTROL2 DRIVE TYPE A.
- #define **SDHC\_CTRL2\_DRV\_TYPE\_C** (0x0020U)  
SDHC HOST CONTROL2 DRIVE TYPE C.
- #define **SDHC\_CTRL2\_DRV\_TYPE\_D** (0x0030U)  
SDHC HOST CONTROL2 DRIVE TYPE D.
- #define **SDHC\_CTRL2\_EXEC\_TUNING** (0x0040U)  
SDHC HOST CONTROL2 EXECUTE TUNING.

## SDHC Standard Definition

- #define **SDHC\_CTRL2\_TUNED\_CLK** (0x0080U)  
*SDHC HOST CONTROL2 SAMPLING CLOCK SELECT.*
- #define **SDHC\_CTRL2\_ASNYC\_INTR\_EN** (0x4000U)  
*SDHC HOST CONTROL2 ASYNC INTERRUPT ENABLE.*
- #define **SDHC\_CTRL2\_PRESET\_VAL\_EN** (0x8000U)  
*SDHC HOST CONTROL2 PRESET VALUE ENABLE.*
- #define **SDHC\_HOST\_CAPABILITIES** (0x40U)  
*SDHC CAPABILITIES REG.*
- #define **SDHC\_HCAP\_TOCLKFREQ\_MASK** (0x0000003F)  
*SDHC CAPABILITIES TIMEOUT CLOCK FREQUENCY.*
- #define **SDHC\_HCAP\_TOCKLUINT\_MHZ** (0x00000080U)  
*SDHC CAPABILITIES TIMEOUT CLOCK UNIT.*
- #define **SDHC\_HCAP\_CLK\_BASE\_MASK** (0x00003F00U)  
*SDHC CAPABILITIES BASE CLOCK FREQUENCY FOR SD CLOCK MASK.*
- #define **SDHC\_HCAP\_MAX\_BLK\_LSF** (16U)  
*SDHC CAPABILITIES MAX BLOCK LENGTH SHIFT.*
- #define **SDHC\_HCAP\_MAX\_BLK\_MASK** (0x00030000U)  
*SDHC CAPABILITIES MAX BLOCK LENGTH MASK.*
- #define **SDHC\_HCAP\_MAXBLK\_512** (0x0U)  
*SDHC CAPABILITIES MAX BLOCK LENGTH 512B.*
- #define **SDHC\_HCAP\_MAXBLK\_1024** (0x1U)  
*SDHC CAPABILITIES MAX BLOCK LENGTH 1024B.*
- #define **SDHC\_HCAP\_MAXBLK\_2048** (0x2U)  
*SDHC CAPABILITIES MAX BLOCK LENGTH 2048B.*
- #define **SDHC\_HCAP\_SUPPORT\_8BIT** (0x00040000U)  
*SDHC CAPABILITIES SUPPORT 8 BIT.*
- #define **SDHC\_HCAP\_SUPPORT\_ADMA2** (0x00080000U)  
*SDHC CAPABILITIES SUPPORT ADMA2.*
- #define **SDHC\_HCAP\_SUPPORT\_ADMA1** (0x00100000U)  
*SDHC CAPABILITIES SUPPORT ADMA1.*
- #define **SDHC\_HCAP\_SUPPORT\_HISPD** (0x00200000U)  
*SDHC CAPABILITIES SUPPORT HIGH SPEED.*
- #define **SDHC\_HCAP\_SUPPORT\_SDMA** (0x00400000U)  
*SDHC CAPABILITIES SUPPORT SDMA.*
- #define **SDHC\_HCAP\_SUPPORT\_SUSPEND** (0x00800000U)  
*SDHC CAPABILITIES SUPPORT SUSPEND RESUME.*
- #define **SDHC\_HCAP\_SUPPORT\_V330** (0x01000000U)  
*SDHC CAPABILITIES SUPPORT 3.3V.*
- #define **SDHC\_HCAP\_SUPPORT\_V300** (0x02000000U)  
*SDHC CAPABILITIES SUPPORT 3.0V.*
- #define **SDHC\_HCAP\_SUPPORT\_V180** (0x04000000U)  
*SDHC CAPABILITIES SUPPORT 1.8V.*
- #define **SDHC\_HCAP\_SUPPORT\_64BIT** (0x10000000U)  
*SDHC CAPABILITIES SUPPORT 64-BIT.*
- #define **SDHC\_HCAP\_SUPPORT\_ASYNC** (0x20000000U)  
*SDHC CAPABILITIES SUPPORT ASYNC INTERRUPT.*
- #define **SDHC\_HCAP\_SLOT\_TYPE\_LSF** (30U)  
*SDHC CAPABILITIES SLOT TYPE SHIFT.*
- #define **SDHC\_HCAP\_SLOT\_TYPE\_MASK** (0xC0000000U)  
*SDHC CAPABILITIES SLOT TYPE MASK.*
- #define **SDHC\_HCAP\_SLOT\_REMOVABLE** (0x0U)

- #define SDHC\_HCACP\_SLOT\_EMBEDDED (0x1U)  
SDHC CAPABILITIES SLOT TYPE EMBEDDED.
- #define SDHC\_HCACP\_SLOT\_SHARED (0x2U)  
SDHC CAPABILITIES SLOT TYPE SHARED BUS SLOT.
- #define SDHC\_HOST\_CAPABILITIES\_1 (0x44U)  
SDHC CAPABILITIES1 REG.
- #define SDHC\_HCACP\_SUPPORT\_SDR50 (0x00000001U)  
SDHC CAPABILITIES1 SUPPORT SDR50.
- #define SDHC\_HCACP\_SUPPORT\_SDR104 (0x00000002U)  
SDHC CAPABILITIES1 SUPPORT SDR104.
- #define SDHC\_HCACP\_SUPPORT\_DDR50 (0x00000004U)  
SDHC CAPABILITIES1 SUPPORT DDR50.
- #define SDHC\_HCACP\_DRIVER\_TYPE\_A (0x00000010U)  
SDHC CAPABILITIES1 SUPPORT DRIVER TYPE A.
- #define SDHC\_HCACP\_DRIVER\_TYPE\_C (0x00000020U)  
SDHC CAPABILITIES1 SUPPORT DRIVER TYPE C.
- #define SDHC\_HCACP\_DRIVER\_TYPE\_D (0x00000040U)  
SDHC CAPABILITIES1 SUPPORT DRIVER TYPE D.
- #define SDHC\_HCACP\_RT\_TMCNT\_LSF (8U)  
SDHC CAPABILITIES1 TIMER COUNT FOR RETUNING SHIFT.
- #define SDHC\_HCACP\_RT\_TMCNT\_MASK (0x00000F00U)  
SDHC CAPABILITIES1 TIMER COUNT FOR RETUNING MASK.
- #define SDHC\_HCACP\_USE\_SDR50\_TUNE (0x00002000U)  
SDHC CAPABILITIES1 USE TUNING FOR SDR50.
- #define SDHC\_HCACP\_RT\_MODE\_LSF (14U)  
SDHC CAPABILITIES1 RETUNE MODE SHIFT.
- #define SDHC\_HCACP\_RT\_MODE\_MASK (0x0000C000U)  
SDHC CAPABILITIES1 RETUNE MODE MASK.
- #define SDHC\_HCACP\_CLK\_MUL\_LSF (16U)  
SDHC CAPABILITIES1 CLOCK MULTIPLIER SHIFT.
- #define SDHC\_HCACP\_CLK\_MUL\_MASK (0x00FF0000U)  
SDHC CAPABILITIES1 CLOCK MULTIPLIER MASK.
- #define SDHC\_MAX\_CURRENT (0x48U)  
SDHC MAX CURRENT REG.
- #define SDHC\_MC\_330\_LSF (0U)  
SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.3V SHIFT.
- #define SDHC\_MC\_330\_MASK (0x0000FF)  
SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.3V MASK.
- #define SDHC\_MC\_300\_LSF (8U)  
SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.0V SHIFT.
- #define SDHC\_MC\_300\_MASK (0x00FF00U)  
SDHC MAX CURRENT MAXIMUM CURRENT FOR 3.0V MASK.
- #define SDHC\_MC\_180\_LSF (16U)  
SDHC MAX CURRENT MAXIMUM CURRENT FOR 1.8V SHIFT.
- #define SDHC\_MC\_180\_MASK (0xFF0000U)  
SDHC MAX CURRENT MAXIMUM CURRENT FOR 1.8V MASK.
- #define SDHC\_FRC\_EVENT\_AUTOCMD (0x50U)  
SDHC FORCE EVENT FOR AUTOCMD REG.
- #define SDHC\_FEA\_E\_NO\_ACMD12\_EXEC (0x0001U)  
SDHC FORCE EVENT AUTO CMD12 NOT EXECUTED.

## SDHC Standard Definition

- #define **SDHC\_FEA\_E\_ACMD\_TIMEOUT** (0x002U)  
*SDHC FORCE EVENT AUTO CMD TIMEOUT ERROR.*
- #define **SDHC\_FEA\_E\_ACMD\_CRC** (0x0004U)  
*SDHC FORCE EVENT AUTO CMD CRC ERROR.*
- #define **SDHC\_FEA\_E\_ACMD\_END** (0x0008U)  
*SDHC FORCE EVENT AUTO CMD END BIT ERROR.*
- #define **SDHC\_FEA\_E\_ACMD\_INDEX** (0x0010U)  
*SDHC FORCE EVENT AUTO CMD INDEX ERROR.*
- #define **SDHC\_FEA\_E\_CMD\_NOT\_BY\_ACMD12** (0x0080U)  
*SDHC FORCE EVENT AUTO CMD NOT ISSUED ERROR.*
- #define **SDHC\_FRC\_EVENT\_ERROR\_INTR** (0x52U)  
*SDHC FORCE EVENT FOR ERROR REG.*
- #define **SDHC\_FEI\_E\_CMD\_TIMEOUT** (0x0001U)  
*SDHC FORCE CMD TIMEOUT ERROR.*
- #define **SDHC\_FEI\_E\_CMD\_CRC** (0x0002U)  
*SDHC FORCE CMD CRC ERROR.*
- #define **SDHC\_FEI\_E\_CMD\_END\_BIT** (0x0004U)  
*SDHC FORCE CMD END BIT ERROR.*
- #define **SDHC\_FEI\_E\_DATA\_TIMEOUT** (0x0008U)  
*SDHC FORCE DATA TIMEOUT ERROR.*
- #define **SDHC\_FEI\_E\_DATA\_CRC** (0x0010U)  
*SDHC FORCE DATA CRC ERROR.*
- #define **SDHC\_FEI\_E\_DATA\_END\_BIT** (0x0020U)  
*SDHC FORCE DATA END BIT ERROR.*
- #define **SDHC\_FEI\_E\_CURRENT\_LIMIT** (0x0040U)  
*SDHC FORCE CURRENT LIMIT ERROR.*
- #define **SDHC\_FEI\_E\_AUTO\_CMD** (0x0080U)  
*SDHC FORCE AUTOCMD ERROR.*
- #define **SDHC\_FEI\_E\_ADMA** (0x0100U)  
*SDHC FORCE ADMA ERROR.*
- #define **SDHC\_ADMA\_ERROR** (0x54U)  
*SDHC ADMA ERROR REG.*
- #define **SDHC\_ADMA\_ADDRESS** (0x58U)  
*SDHC ADMA ADDRESS REG.*
- #define **SDHC\_SLOT\_INT\_STATUS** (0xFCU)  
*SDHC SLOT INTERRUPT STATUS REG.*
- #define **SDHC\_HOST\_VERSION** (0xFEU)  
*SDHC HOST CONTROLLER VERSION REG.*
- #define **SDHC\_VENDOR\_VER\_LSF** (8U)  
*SDHC HOST CONTROLLER VERSION VENDOR VERSION SHIFT.*
- #define **SDHC\_VENDOR\_VER\_MASK** (0xFF00U)  
*SDHC HOST CONTROLLER VERSION VENDOR VERSION MASK.*
- #define **SDHC\_SPEC\_VER\_LSF** (0U)  
*SDHC HOST CONTROLLER VERSION SPEC VERSION SHIFT.*
- #define **SDHC\_SPEC\_VER\_MASK** (0x00FFU)  
*SDHC HOST CONTROLLER VERSION SPEC VERSION MASK.*
- #define **SDHC\_SPEC\_100** (0U)  
*SDHC HOST CONTROLLER VERSION SPEC VERSION 1.00.*
- #define **SDHC\_SPEC\_200** (1U)  
*SDHC HOST CONTROLLER VERSION SPEC VERSION 2.00.*
- #define **SDHC\_SPEC\_300** (2U)

*SDHC HOST CONTROLLER VERSION SPEC VERSION 3.00.*

## SDHC Card Related Standard Definition

### 27.6 SDHC Card Related Standard Definition

#### 27.6.1 Overview

This section describes the card standard definition.

#### Macros

- #define **SDMMC\_CARD\_BUSY** ((uint32\_t) 1 << 31)  
*card initialization complete*
- #define **SD\_SCR\_BUS\_WIDTHS\_1BIT** (1 << 0)  
*card supports 1 bit mode*
- #define **SD\_SCR\_BUS\_WIDTHS\_4BIT** (1 << 2)  
*card supports 4 bit mode*
- #define **SD\_CCC\_BASIC** (1 << 0)  
*Card command class 0.*
- #define **SD\_CCC\_BLOCK\_READ** (1 << 2)  
*Card command class 2.*
- #define **SD\_CCC\_BLOCK\_WRITE** (1 << 4)  
*Card command class 4.*
- #define **SD\_CCC\_ERASE** (1 << 5)  
*Card command class 5.*
- #define **SD\_CCC\_WRITE\_PROTECTION** (1 << 6)  
*Card command class 6.*
- #define **SD\_CCC\_LOCK\_CARD** (1 << 7)  
*Card command class 7.*
- #define **SD\_CCC\_APP\_SPEC** (1 << 8)  
*Card command class 8.*
- #define **SD\_CCC\_IO\_MODE** (1 << 9)  
*Card command class 9.*
- #define **SD\_CCC\_SWITCH** (1 << 10)  
*Card command class 10.*
- #define **SD\_OCR\_CCS** (1 << 30)  
*card capacity status*
- #define **SD\_OCR\_HCS** (1 << 30)  
*card capacity status*
- #define **SD\_OCR\_XPC** (1 << 28)  
*SDXC power control.*
- #define **SD\_OCR\_S18R** (1 << 24)  
*switch to 1.8V request*
- #define **SD\_OCR\_S18A SD\_OCR\_S18R**  
*switch to 1.8V accepted*
- #define **SD\_HIGHSPEED\_BUSY** (0x00020000U)  
*SD card high speed busy status bit in CMD6 response.*
- #define **SD\_HIGHSPEED\_SUPPORTED** (0x00020000U)  
*SD card high speed support bit in CMD6 response.*
- #define **SD\_OCR\_VDD\_27\_28** (1 << 15)  
*VDD 2.7-2.8.*
- #define **SD\_OCR\_VDD\_28\_29** (1 << 16)  
*VDD 2.8-2.9.*

- #define **SD\_OCR\_VDD\_29\_30** (1 << 17)  
*VDD 2.9-3.0.*
- #define **SD\_OCR\_VDD\_30\_31** (1 << 18)  
*VDD 3.0-3.1.*
- #define **SD\_OCR\_VDD\_31\_32** (1 << 19)  
*VDD 3.1-3.2.*
- #define **SD\_OCR\_VDD\_32\_33** (1 << 20)  
*VDD 3.2-3.3.*
- #define **SD\_OCR\_VDD\_33\_34** (1 << 21)  
*VDD 3.3-3.4.*
- #define **SD\_OCR\_VDD\_34\_35** (1 << 22)  
*VDD 3.4-3.5.*
- #define **SD\_OCR\_VDD\_35\_36** (1 << 23)  
*VDD 3.5-3.6.*
- #define **SDMMC\_R1\_OUT\_OF\_RANGE** ((uint32\_t) 1 << 31)  
*R1: out of range status bit.*
- #define **SDMMC\_R1\_ADDRESS\_ERROR** (1 << 30)  
*R1: address error status bit.*
- #define **SDMMC\_R1\_BLOCK\_LEN\_ERROR** (1 << 29)  
*R1: block length error status bit.*
- #define **SDMMC\_R1\_ERASE\_SEQ\_ERROR** (1 << 28)  
*R1: erase sequence error status bit.*
- #define **SDMMC\_R1\_ERASE\_PARAM** (1 << 27)  
*R1: erase parameter error status bit.*
- #define **SDMMC\_R1\_WP\_VIOLATION** (1 << 26)  
*R1: write protection violation status bit.*
- #define **SDMMC\_R1\_CARD\_IS\_LOCKED** (1 << 25)  
*R1: card locked status bit.*
- #define **SDMMC\_R1\_LOCK\_UNLOCK\_FAILED** (1 << 24)  
*R1: lock/unlock error status bit.*
- #define **SDMMC\_R1\_COM\_CRC\_ERROR** (1 << 23)  
*R1: CRC error status bit.*
- #define **SDMMC\_R1\_ILLEGAL\_COMMAND** (1 << 22)  
*R1: illegal command status bit.*
- #define **SDMMC\_R1\_CARD\_ECC\_FAILED** (1 << 21)  
*R1: card ecc error status bit.*
- #define **SDMMC\_R1\_CC\_ERROR** (1 << 20)  
*R1: internal card controller status bit.*
- #define **SDMMC\_R1\_ERROR** (1 << 19)  
*R1: a general or an unknown error status bit.*
- #define **SDMMC\_R1\_CID\_CSD\_OVERWRITE** (1 << 16)  
*R1: cid/csd overwrite status bit.*
- #define **SDMMC\_R1\_WP\_ERASE\_SKIP** (1 << 15)  
*R1: write protection erase skip status bit.*
- #define **SDMMC\_R1\_CARD\_ECC\_DISABLED** (1 << 14)  
*R1: card ecc disabled status bit.*
- #define **SDMMC\_R1\_ERASE\_RESET** (1 << 13)  
*R1: erase reset status bit.*
- #define **SDMMC\_R1\_STATUS**(x) ((uint32\_t)(x) & 0xFFFFE000U)  
*R1: status.*
- #define **SDMMC\_R1\_READY\_FOR\_DATA** (1 << 8)

## SDHC Card Related Standard Definition

- `#define SDMMC_R1_SWITCH_ERROR (1 << 7)`  
*R1: ready for data status bit.*
- `#define SDMMC_R1_APP_CMD (1 << 5)`  
*R1: switch error status bit.*
- `#define SDMMC_R1_AKE_SEQ_ERROR (1 << 3)`  
*R1: application command enabled status bit.*
- `#define SDMMC_R1_ERROR_BITS(x)`  
*R1: error in the sequence of the authentication process.*
- `#define SDMMC_R1_CURRENT_STATE(x) (((x) & 0x00001E00U) >> 9)`  
*R1: current state.*
- `#define SDMMC_R1_STATE_IDLE (0U)`  
*R1: current state: idle.*
- `#define SDMMC_R1_STATE_READY (1U)`  
*R1: current state: ready.*
- `#define SDMMC_R1_STATE_IDENT (2U)`  
*R1: current state: ident.*
- `#define SDMMC_R1_STATE_STBY (3U)`  
*R1: current state: stby.*
- `#define SDMMC_R1_STATE_TRAN (4U)`  
*R1: current state: tran.*
- `#define SDMMC_R1_STATE_DATA (5U)`  
*R1: current state: data.*
- `#define SDMMC_R1_STATE_RCV (6U)`  
*R1: current state: rcv.*
- `#define SDMMC_R1_STATE_PRG (7U)`  
*R1: current state: prg.*
- `#define SDMMC_R1_STATE_DIS (8U)`  
*R1: current state: dis.*
- `#define SDMMC_SD_VERSION_1_0 (1 << 0)`  
*SD card version 1.0.*
- `#define SDMMC_SD_VERSION_1_1 (1 << 1)`  
*SD card version 1.1.*
- `#define SDMMC_SD_VERSION_2_0 (1 << 2)`  
*SD card version 2.0.*
- `#define SDMMC_SD_VERSION_3_0 (1 << 3)`  
*SD card version 3.0.*
- `#define SDMMC_SPI_DET_ERROR (1 << 0)`  
*Data error.*
- `#define SDMMC_SPI_DET_CC_ERROR (1 << 1)`  
*CC error.*
- `#define SDMMC_SPI_DET_ECC_FAILED (1 << 2)`  
*Card ecc error.*
- `#define SDMMC_SPI_DET_OUT_OF_RANGE (1 << 3)`  
*Out of range.*
- `#define SDMMC_SPI_DT_START_SINGLE_BLK (0xFEU)`  
*First byte of block, single block.*
- `#define SDMMC_SPI_DT_START_MULTI_BLK (0xFCU)`  
*First byte of block, multi-block.*
- `#define SDMMC_SPI_DT_STOP_TRANSFER (0xFDU)`  
*Stop transmission.*

- #define **SDMMC\_SPI\_DR\_MASK** (0x1F)  
*Mask for data response bits.*
- #define **SDMMC\_SPI\_DR\_ACCEPTED** (0x05)  
*Data accepted.*
- #define **SDMMC\_SPI\_DR\_CRC\_ERROR** (0x0B)  
*Data rejected due to CRC error.*
- #define **SDMMC\_SPI\_DR\_WRITE\_ERROR** (0x0D)  
*Data rejected due to write error.*

## Enumerations

- enum **mmc\_cmd\_t** {  
  kMmcSetRelativeAddr = 3,  
  kMmcSleepAwake = 5,  
  kMmcSwitch = 6,  
  kMmcSendExtCsd = 8,  
  kMmcReadDataUntilStop = 11,  
  kMmcBusTestRead = 14,  
  kMmcWriteDataUntilStop = 20,  
  kMmcProgramCid = 26,  
  kMmcEraseGroupStart = 35,  
  kMmcEraseGroupEnd = 36,  
  kMmcFastIo = 39,  
  kMmcGoIrqState = 40 }
- enum **sdmmc\_cmd\_t** {

## SDHC Card Related Standard Definition

```
kGoIdleState = 0,
kSendOpCond = 1,
kAllSendCid = 2,
kSetDsr = 4,
kSelectCard = 7,
kSendCsd = 9,
kSendCid = 10,
kStopTransmission = 12,
kSendStatus = 13,
kGoInactiveState = 15,
kSetBlockLen = 16,
kReadSingleBlock = 17,
kReadMultipleBlock = 18,
kSendTuningBlock = 19,
kSetBlockCount = 23,
kWriteBlock = 24,
kWriteMultipleBlock = 25,
kProgramCsd = 27,
kSetWriteProt = 28,
kClrWriteProt = 29,
kSendWriteProt = 30,
kErase = 38,
kLockUnlock = 42,
kAppCmd = 55,
kGenCmd = 56 }
• enum sd_cmd_t {
 kSdSendRelativeAddr = 3,
 kSdSwitch = 6,
 kSdSendIfCond = 8,
 kSdVoltageSwitch = 11,
 kSdSpeedClassControl = 20,
 kSdEraseWrBlkStart = 32,
 kSdEraseWrBlkEnd = 33 }
• enum sd_acmd_t {
 kSdAppSetBusWdith = 6,
 kSdAppStatus = 13,
 kSdAppSendNumWrBlocks = 22,
 kSdAppSetWrBlkEraseCount = 23,
 kSdAppSendOpCond = 41,
 kSdAppSetClrCardDetect = 42,
 kSdAppSendScr = 51 }
• enum sd_buswidth_t {
 kSdBusWidth1Bit = 0,
 kSdBusWidth4Bit = 2 }
• enum sd_switch_mode_t {
```

```

kSdSwitchCheck = 0,
kSdSwitchSet = 1 }
• enum sdcards_type_t {
 kCardTypeUnknown = 1,
 kCardTypeSd,
 kCardTypeMmc,
 kCardTypeSdio }

```

## 27.6.2 Enumeration Type Documentation

### 27.6.2.1 enum mmc\_cmd\_t

Enumerator

```

kMmcSetRelativeAddr ac [31:16] RCA R1
kMmcSleepAwake ac [31:16] RCA R1b [15] flag
kMmcSwitch ac [31:16] RCA R1b
kMmcSendExtCsd adtc R1
kMmcReadDataUntilStop adtc [31:0] data R1 address
kMmcBusTestRead adtc R1
kMmcWriteDataUntilStop ac [31:0] data R1 address
kMmcProgramCid adtc R1
kMmcEraseGroupStart ac [31:0] data R1 address
kMmcEraseGroupEnd ac [31:0] data R1 address
kMmcFastIo ac R4
kMmcGoIrqState bcr R5

```

### 27.6.2.2 enum sdmmc\_cmd\_t

Enumerator

```

kGoIdleState bc
kSendOpCond bcr [31:0] OCR R3
kAllSendCid bcr R2
kSetDsr bc [31:16] RCA
kSelectCard ac [31:16] RCA R1b
kSendCsd ac [31:16] RCA R2
kSendCid ac [31:16] RCA R2
kStopTransmission ac [31:16] RCA R1b
kSendStatus ac [31:16] RCA R1
kGoInactiveState ac [31:16] RCA
kSetBlockLen ac [31:0] block R1 length
kReadSingleBlock adtc [31:0] data R1 address
kReadMultipleBlock adtc [31:0] data R1 address

```

## SDHC Card Related Standard Definition

*kSendTuningBlock* adtc [31:0] all R1 zero  
*kSetBlockCount* ac [31:0] block R1 count  
*kWriteBlock* adtc [31:0] data R1 address  
*kWriteMultipleBlock* adtc [31:0] data R1 address  
*kProgramCsd* adtc R1  
*kSetWriteProt* ac [31:0] data R1b address  
*kClrWriteProt* ac [31:0] data R1b address  
*kSendWriteProt* adtc [31:0] write R1b protect data address  
*kErase* ac R1  
*kLockUnlock* adtc all zero R1  
*kAppCmd* ac [31:16] RCA R1  
*kGenCmd* adtc [0] RD/WR R1

### 27.6.2.3 enum sd\_cmd\_t

Enumerator

*kSdSendRelativeAddr* bcr R6  
*kSdSwitch* adtc [31] mode R1 [15:12] func group 4: current limit [11:8] func group 3: drive strength [7:4] func group 2: command system [3:0] func group 1: access mode  
*kSdSendIfCond* bcr [11:8] supply R7 voltage [7:0] check pattern  
*kSdVoltageSwitch* ac R1  
*kSdSpeedClassControl* ac [31:28] speed R1b class control  
*kSdEraseWrBlkStart* ac [31:0] data R1 address  
*kSdEraseWrBlkEnd* ac [31:0] data R1 address

### 27.6.2.4 enum sd\_acmd\_t

Enumerator

*kSdAppSetBusWidth* ac [1:0] bus R1 width  
*kSdAppStatus* adtc R1  
*kSdAppSendNumWrBlocks* adtc R1  
*kSdAppSetWrBlkEraseCount* ac [22:0] number R1 of blocks  
*kSdAppSendOpCond* bcr [30] HCS R3 [28] XPC [24] S18R [23:0] VDD voltage window  
*kSdAppSetClrCardDetect* ac [0] set cd R1  
*kSdAppSendScr* adtc R1

### 27.6.2.5 enum sd\_buswidth\_t

Enumerator

*kSdBusWidth1Bit* SD data bus width 1-bit mode.  
*kSdBusWidth4Bit* SD data bus width 1-bit mode.

### **27.6.2.6 enum sd\_switch\_mode\_t**

Enumerator

*kSdSwitchCheck* SD switch mode 0: check function.

*kSdSwitchSet* SD switch mode 1: set function.

### **27.6.2.7 enum sdcard\_type\_t**

Enumerator

*kCardTypeUnknown* Unknown card type.

*kCardTypeSd* SD card type.

*kCardTypeMmc* MMC card type.

*kCardTypeSdio* SDIO card type.

### 27.7 SDHC Card Definition

This chapter describes the programming interface of the card data definition for FSL SD host controller.

## 27.8 SDHC Card Driver

This chapter describes the programming interface of the card driver for FSL SD host controller.

### 27.9 SD Card SPI Data Definition

The chapter describes the programming interface of the card data definition over SPI.

## **27.10 SD Card SPI Driver**

The chapter describes the programming interface of the card driver over SPI.



# Chapter 28

## Serial Peripheral Interface (SPI)

### 28.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Serial Peripheral Interface (SPI) block of Kinetis L-series devices. For K- and V-series Kinetis devices, refer to the DSPI module driver.

### Modules

- [SPI Classes](#)
- [SPI HAL driver](#)
- [SPI Master Peripheral Driver](#)
- [SPI Slave Peripheral Driver](#)
- [Shared SPI Types](#)

### 28.2 SPI HAL driver

#### 28.2.1 Overview

This section describes the programming interface of the SPI HAL driver.

## Files

- file [fsl\\_spi\\_hal.h](#)

## Enumerations

- enum `spi_status_t` {  
  `kStatus_SPI_SlaveTxUnderrun`,  
  `kStatus_SPI_SlaveRxOverrun`,  
  `kStatus_SPI_Timeout`,  
  `kStatus_SPI_Busy`,  
  `kStatus_SPI_NoTransferInProgress`,  
  `kStatus_SPI_OutOfRange`,  
  `kStatus_SPI_TxBufferNotEmpty`,  
  `kStatus_SPI_InvalidParameter`,  
  `kStatus_SPI_NonInit`,  
  `kStatus_SPI_AlreadyInitialized`,  
  `kStatus_SPI_DMAChannelInvalid` }  
    *Error codes for the SPI driver.*
- enum `spi_master_slave_mode_t` {  
  `kSpiMaster` = 1,  
  `kSpiSlave` = 0 }  
    *SPI master or slave configuration.*
- enum `spi_clock_polarity_t` {  
  `kSpiClockPolarity_ActiveHigh` = 0,  
  `kSpiClockPolarity_ActiveLow` = 1 }  
    *SPI clock polarity configuration.*
- enum `spi_clock_phase_t` {  
  `kSpiClockPhase_FirstEdge` = 0,  
  `kSpiClockPhase_SecondEdge` = 1 }  
    *SPI clock phase configuration.*
- enum `spi_shift_direction_t` {  
  `kSpiMsbFirst` = 0,  
  `kSpiLsbFirst` = 1 }  
    *SPI data shifter direction options.*
- enum `spi_ss_output_mode_t` {  
  `kSpiSlaveSelect_AsGpio` = 0,  
  `kSpiSlaveSelect_FaultInput` = 2,  
  `kSpiSlaveSelect_AutomaticOutput` = 3 }

- *SPI slave select output mode options.*
- enum `spi_pin_mode_t` {
   
    `kSpiPinMode_Normal` = 0,
   
    `kSpiPinMode_Input` = 1,
   
    `kSpiPinMode_Output` = 3 }
   
*SPI pin mode options.*
- enum `spi_data_bitcount_mode_t` {
   
    `kSpi8BitMode` = 0,
   
    `kSpi16BitMode` = 1 }
   
*SPI data length mode options.*
- enum `spi_fifo_interrupt_source_t` {
   
    `kSpiRx_fifoNearFullInt` = 1,
   
    `kSpiTx_fifoNearEmptyInt` = 2 }
   
*SPI FIFO interrupt sources.*
- enum `spi_w1c_interrupt_t` {
   
    `kSpiRx_fifoFullClearInt` = 0,
   
    `kSpiTx_fifoEmptyClearInt` = 1,
   
    `kSpiRx_nearFullClearInt` = 2,
   
    `kSpiTx_nearEmptyClearInt` = 3 }
   
*SPI FIFO write-1-to-clear interrupt flags.*
- enum `spi_txfifo_watermark_t`
  
*SPI TX FIFO watermark settings.*
- enum `spi_rxfifo_watermark_t`
  
*SPI RX FIFO watermark settings.*
- enum `spi_fifo_status_flag_t`
  
*SPI status flags.*
- enum `spi_fifo_error_flag_t` {
   
    `kSpiNoFifoError` = 0,
   
    `kSpiRxfifof` = 1,
   
    `kSpiTxfifof` = 2,
   
    `kSpiRxfifofTxfifof` = 3,
   
    `kSpiRxferr` = 4,
   
    `kSpiRxfifofRxferr` = 5,
   
    `kSpiTxfifofRxferr` = 6,
   
    `kSpiRxfifofTxfifofRxferr` = 7,
   
    `kSpiTxferr` = 8,
   
    `kSpiRxfifofTxferr` = 9,
   
    `kSpiTxfifofTxferr` = 10,
   
    `kSpiRxfifofTxfifofTxferr` = 11,
   
    `kSpiRxferrTxferr` = 12,
   
    `kSpiRxfifofRxferrTxferr` = 13,
   
    `kSpiTxfifofRxferrTxferr` = 14,
   
    `kSpiRxfifofTxfifofRxferrTxferr` = 15 }
   
*SPI error flags.*

## SPI HAL driver

### Configuration

- void **SPI\_HAL\_Init** (uint32\_t baseAddr)  
*Restores the SPI to reset configuration.*
- static void **SPI\_HAL\_Enable** (uint32\_t baseAddr)  
*Enables the SPI peripheral.*
- static void **SPI\_HAL\_Disable** (uint32\_t baseAddr)  
*Disables the SPI peripheral.*
- uint32\_t **SPI\_HAL\_SetBaud** (uint32\_t baseAddr, uint32\_t bitsPerSec, uint32\_t sourceClockInHz)  
*Sets the SPI baud rate in bits per second.*
- static void **SPI\_HAL\_SetBaudDivisors** (uint32\_t baseAddr, uint32\_t prescaleDivisor, uint32\_t rateDivisor)  
*Configures the baud rate divisors manually.*
- static void **SPI\_HAL\_SetMasterSlave** (uint32\_t baseAddr, **spi\_master\_slave\_mode\_t** mode)  
*Configures the SPI for master or slave.*
- static bool **SPI\_HAL\_IsMaster** (uint32\_t baseAddr)  
*Returns whether the SPI module is in master mode.*
- void **SPI\_HAL\_SetSlaveSelectOutputMode** (uint32\_t baseAddr, **spi\_ss\_output\_mode\_t** mode)  
*Sets how the slave select output operates.*
- void **SPI\_HAL\_SetDataFormat** (uint32\_t baseAddr, **spi\_clock\_polarity\_t** polarity, **spi\_clock\_phase\_t** phase, **spi\_shift\_direction\_t** direction)  
*Sets the polarity, phase, and shift direction.*
- static uint32\_t **SPI\_HAL\_GetDataRegAddr** (uint32\_t baseAddr)  
*Gets the SPI data register address for DMA operation.*
- void **SPI\_HAL\_SetPinMode** (uint32\_t baseAddr, **spi\_pin\_mode\_t** mode)  
*Sets the SPI pin mode.*

### Low power

- static void **SPI\_HAL\_ConfigureStopInWaitMode** (uint32\_t baseAddr, bool enable)  
*Enables or disables the SPI clock to stop when the CPU enters wait mode.*

### Interrupts

- static void **SPI\_HAL\_SetReceiveAndFaultIntCmd** (uint32\_t baseAddr, bool enable)  
*Enables or disables the SPI receive buffer/FIFO full and mode fault interrupt.*
- static bool **SPI\_HAL\_GetReceiveAndFaultIntCmd** (uint32\_t baseAddr)  
*Returns the SPI receive buffer/FIFO full and mode fault interrupt setting.*
- static void **SPI\_HAL\_SetTransmitIntCmd** (uint32\_t baseAddr, bool enable)  
*Enables or disables the SPI transmit buffer/FIFO empty interrupt.*
- static bool **SPI\_HAL\_GetTransmitIntCmd** (uint32\_t baseAddr)  
*Returns the SPI transmit buffer/FIFO empty interrupt setting.*
- static void **SPI\_HAL\_SetMatchIntCmd** (uint32\_t baseAddr, bool enable)  
*Enables or disables the SPI match interrupt.*
- static bool **SPI\_HAL\_GetMatchIntCmd** (uint32\_t baseAddr)  
*Returns the SPI match interrupt setting.*

## Status

- static bool **SPI\_HAL\_IsReadBuffFullPending** (uint32\_t baseAddr)  
*Checks whether the read buffer/FIFO is full.*
- static bool **SPI\_HAL\_IsTxBuffEmptyPending** (uint32\_t baseAddr)  
*Checks whether the transmit buffer/FIFO is empty.*
- static bool **SPI\_HAL\_IsModeFaultPending** (uint32\_t baseAddr)  
*Checks whether a mode fault occurred.*
- void **SPI\_HAL\_ClearModeFaultFlag** (uint32\_t baseAddr)  
*Clears the mode fault flag.*
- static bool **SPI\_HAL\_IsMatchPending** (uint32\_t baseAddr)  
*Checks whether the data received matches the previously-set match value.*
- void **SPI\_HAL\_ClearMatchFlag** (uint32\_t baseAddr)  
*Clears the match flag.*

## Data transfer

- static uint8\_t **SPI\_HAL\_ReadData** (uint32\_t baseAddr)  
*Reads a byte from the data buffer.*
- static void **SPI\_HAL\_WriteData** (uint32\_t baseAddr, uint8\_t data)  
*Writes a byte into the data buffer.*
- void **SPI\_HAL\_WriteDataBlocking** (uint32\_t baseAddr, uint8\_t data)  
*Writes a byte into the data buffer and waits till complete to return.*

## Match byte

- static void **SPI\_HAL\_SetMatchValue** (uint32\_t baseAddr, uint8\_t matchByte)  
*Sets the value which triggers the match interrupt.*

### 28.2.2 Enumeration Type Documentation

#### 28.2.2.1 enum spi\_status\_t

Enumerator

- kStatus\_SPI\_SlaveTxUnderrun** SPI Slave TX Underrun error.
- kStatus\_SPI\_SlaveRxOverrun** SPI Slave RX Overrun error.
- kStatus\_SPI\_Timeout** SPI transfer timed out.
- kStatus\_SPI\_Busy** SPI instance is already busy performing a transfer.
- kStatus\_SPI\_NoTransferInProgress** Attempt to abort a transfer when no transfer was in progress.
- kStatus\_SPI\_OutOfRange** SPI out-of-range error used in slave callback.
- kStatus\_SPI\_TxBufferNotEmpty** SPI TX buffer register is not empty.
- kStatus\_SPI\_InvalidParameter** Parameter is invalid.
- kStatus\_SPI\_NonInit** SPI driver is not initialized.
- kStatus\_SPI\_AlreadyInitialized** SPI driver already initialized.

*kStatus\_SPI\_DMACHannelInvalid* SPI driver cannot requests DMA channel.

### 28.2.2.2 enum spi\_master\_slave\_mode\_t

Enumerator

*kSpiMaster* SPI peripheral operates in master mode.

*kSpiSlave* SPI peripheral operates in slave mode.

### 28.2.2.3 enum spi\_clock\_polarity\_t

Enumerator

*kSpiClockPolarity\_ActiveHigh* Active-high SPI clock (idles low).

*kSpiClockPolarity\_ActiveLow* Active-low SPI clock (idles high).

### 28.2.2.4 enum spi\_clock\_phase\_t

Enumerator

*kSpiClockPhase\_FirstEdge* First edge on SPSCK occurs at the middle of the first cycle of a data transfer.

*kSpiClockPhase\_SecondEdge* First edge on SPSCK occurs at the start of the first cycle of a data transfer.

### 28.2.2.5 enum spi\_shift\_direction\_t

Enumerator

*kSpiMsbFirst* Data transfers start with most significant bit.

*kSpiLsbFirst* Data transfers start with least significant bit.

### 28.2.2.6 enum spi\_ss\_output\_mode\_t

Enumerator

*kSpiSlaveSelect\_AsGpio* Slave select pin configured as GPIO.

*kSpiSlaveSelect\_FaultInput* Slave select pin configured for fault detection.

*kSpiSlaveSelect\_AutomaticOutput* Slave select pin configured for automatic SPI output.

**28.2.2.7 enum spi\_pin\_mode\_t**

Enumerator

*kSpiPinMode\_Normal* Pins operate in normal, single-direction mode.*kSpiPinMode\_Input* Bidirectional mode. Master: MOSI pin is input; Slave: MISO pin is input*kSpiPinMode\_Output* Bidirectional mode. Master: MOSI pin is output; Slave: MISO pin is output**28.2.2.8 enum spi\_data\_bitcount\_mode\_t**

Enumerator

*kSpi8BitMode* 8-bit data transmission mode*kSpi16BitMode* 16-bit data transmission mode**28.2.2.9 enum spi\_fifo\_interrupt\_source\_t**

Enumerator

*kSpiRxFifoNearFullInt* Receive FIFO nearly full interrupt.*kSpiTxFifoNearEmptyInt* Transmit FIFO nearly empty interrupt.**28.2.2.10 enum spi\_w1c\_interrupt\_t**

Enumerator

*kSpiRxFifoFullClearInt* Receive FIFO full interrupt.*kSpiTxFifoEmptyClearInt* Transmit FIFO empty interrupt.*kSpiRxNearFullClearInt* Receive FIFO nearly full interrupt.*kSpiTxNearEmptyClearInt* Transmit FIFO nearly empty interrupt.**28.2.2.11 enum spi\_txfifo\_watermark\_t****28.2.2.12 enum spi\_rxfifo\_watermark\_t****28.2.2.13 enum spi\_fifo\_status\_flag\_t****28.2.2.14 enum spi\_fifo\_error\_flag\_t**

Enumerator

*kSpiNoFifoError* No error is detected.*kSpiRxfifo* Rx FIFO Overflow.

## SPI HAL driver

*kSpiTxfof* Tx FIFO Overflow.  
*kSpiRxfofTxfof* Rx FIFO Overflow, Tx FIFO Overflow.  
*kSpiRxferr* Rx FIFO Error.  
*kSpiRxfofRxferr* Rx FIFO Overflow, Rx FIFO Error.  
*kSpiTxfofRxferr* Tx FIFO Overflow, Rx FIFO Error.  
*kSpiRxfofTxfofRxferr* Rx FIFO Overflow, Tx FIFO Overflow, Rx FIFO Error.  
*kSpiTxferr* Tx FIFO Error.  
*kSpiRxfofTxferr* Rx FIFO Overflow, Tx FIFO Error.  
*kSpiTxfofTxferr* Tx FIFO Overflow, Tx FIFO Error.  
*kSpiRxfofTxfofTxferr* Rx FIFO Overflow, Tx FIFO Overflow, Tx FIFO Error.  
*kSpiRxferrTxferr* Rx FIFO Error, Tx FIFO Error.  
*kSpiRxfofRxferrTxferr* Rx FIFO Overflow, Rx FIFO Error, Tx FIFO Error.  
*kSpiTxfofRxferrTxferr* Tx FIFO Overflow, Rx FIFO Error, Tx FIFO Error.  
*kSpiRxfofRxferrTxferrTxferr* Rx FIFO Overflow, Tx FIFO Overflow Rx FIFO Error, Tx FIFO Error.

### 28.2.3 Function Documentation

#### 28.2.3.1 void SPI\_HAL\_Init ( *uint32\_t baseAddr* )

This function basically resets all of the SPI registers to their default setting including disabling the module.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

#### 28.2.3.2 static void SPI\_HAL\_Enable ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

#### 28.2.3.3 static void SPI\_HAL\_Disable ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

#### **28.2.3.4 `uint32_t SPI_HAL_SetBaud ( uint32_t baseAddr, uint32_t bitsPerSec, uint32_t sourceClockInHz )`**

This function takes in the desired bitsPerSec (baud rate) and calculates the nearest possible baud rate without exceeding the desired baud rate unless the baud rate requested is less than the absolute minimum in which case the minimum baud rate will be returned. The returned baud rate is in bits-per-second. It requires that the caller also provide the frequency of the module source clock (in Hertz).

Parameters

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>baseAddr</i>        | Module base address.                      |
| <i>bitsPerSec</i>      | The desired baud rate in bits per second. |
| <i>sourceClockInHz</i> | Module source input clock in Hertz.       |

Returns

The actual calculated baud rate in Hz.

#### **28.2.3.5 `static void SPI_HAL_SetBaudDivisors ( uint32_t baseAddr, uint32_t prescaleDivisor, uint32_t rateDivisor ) [inline], [static]`**

This function allows the caller to manually set the baud rate divisors in the event that these dividers are known and the caller does not wish to call the SPI\_HAL\_SetBaudRate function.

Parameters

|                        |                                     |
|------------------------|-------------------------------------|
| <i>baseAddr</i>        | Module base address.                |
| <i>prescaleDivisor</i> | baud rate prescale divisor setting. |
| <i>rateDivisor</i>     | baud rate divisor setting.          |

#### **28.2.3.6 `static void SPI_HAL_SetMasterSlave ( uint32_t baseAddr, spi_master_slave_mode_t mode ) [inline], [static]`**

Parameters

## SPI HAL driver

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                             |
| <i>mode</i>     | Mode setting (master or slave) of type dspi_master_slave_mode_t. |

### 28.2.3.7 static bool SPI\_HAL\_IsMaster ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

true The module is in master mode. false The module is in slave mode.

### 28.2.3.8 void SPI\_HAL\_SetSlaveSelectOutputMode ( uint32\_t *baseAddr*, spi\_ss\_output\_mode\_t *mode* )

This function allows the user to configure the slave select in one of the three operational modes: as GPIO, as a fault input, or as an automatic output for standard SPI modes.

Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                                |
| <i>mode</i>     | Selection input of one of three modes of type spi_ss_output_mode_t. |

### 28.2.3.9 void SPI\_HAL\_SetDataFormat ( uint32\_t *baseAddr*, spi\_clock\_polarity\_t *polarity*, spi\_clock\_phase\_t *phase*, spi\_shift\_direction\_t *direction* )

This function configures the clock polarity, clock phase, and data shift direction.

Parameters

|                  |                                                                  |
|------------------|------------------------------------------------------------------|
| <i>baseAddr</i>  | Module base address.                                             |
| <i>polarity</i>  | Clock polarity setting of type spi_clock_polarity_t.             |
| <i>phase</i>     | Clock phase setting of type spi_clock_phase_t.                   |
| <i>direction</i> | Data shift direction (MSB or LSB) of type spi_shift_direction_t. |

**28.2.3.10 static uint32\_t SPI\_HAL\_GetDataRegAddr ( uint32\_t *baseAddr* ) [inline],  
[static]**

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

## SPI HAL driver

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

The SPI data register address.

### 28.2.3.11 void SPI\_HAL\_SetPinMode ( uint32\_t *baseAddr*, spi\_pin\_mode\_t *mode* )

This function configures the SPI data pins to one of three modes (of type *spi\_pin\_mode\_t*): Single direction mode: MOSI and MISO pins operate in normal, single direction mode. Bidirectional mode: Master: MOSI configured as input, Slave: MISO configured as input. Bidirectional mode: Master: MOSI configured as output, Slave: MISO configured as output.

Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                    |
| <i>mode</i>     | Operational of SPI pins of type <i>spi_pin_mode_t</i> . |

### 28.2.3.12 static void SPI\_HAL\_ConfigureStopInWaitMode ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function enables or disables the SPI clock operation in wait mode.

Parameters

|                 |                                                              |
|-----------------|--------------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                         |
| <i>enable</i>   | Enable (true) or disable (false) the SPI clock in wait mode. |

### 28.2.3.13 static void SPI\_HAL\_SetReceiveAndFaultIntCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function enables or disables the SPI receive buffer (or FIFO if the module supports a FIFO) full and mode fault interrupt.

Parameters

|                 |                                                                                    |
|-----------------|------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                                               |
| <i>enable</i>   | Enable (true) or disable (false) the receive buffer full and mode fault interrupt. |

#### **28.2.3.14 static bool SPI\_HAL\_GetReceiveAndFaultIntCmd ( uint32\_t *baseAddr* ) [inline], [static]**

This function returns the SPI receive buffer (or FIFO if the module supports a FIFO) full and mode fault interrupt setting.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

The receive buffer full and mode fault interrupt setting.

#### **28.2.3.15 static void SPI\_HAL\_SetTransmitIntCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

This function enables or disables the SPI transmit buffer (or FIFO if the module supports a FIFO) empty interrupt.

Parameters

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                                  |
| <i>enable</i>   | Enable (true) or disable (false) the transmit buffer empty interrupt. |

#### **28.2.3.16 static bool SPI\_HAL\_GetTransmitIntCmd ( uint32\_t *baseAddr* ) [inline], [static]**

This function returns the SPI transmit buffer (or FIFO if the module supports a FIFO) empty interrupt setting.

## SPI HAL driver

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

The transmit buffer empty interrupt setting.

### 28.2.3.17 static void SPI\_HAL\_SetMatchIntCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function enables or disables the SPI match interrupt.

Parameters

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>baseAddr</i> | Module base address.                                  |
| <i>enable</i>   | Enable (true) or disable (false) the match interrupt. |

### 28.2.3.18 static bool SPI\_HAL\_GetMatchIntCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function returns the SPI match interrupt setting.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

The match interrupt setting.

### 28.2.3.19 static bool SPI\_HAL\_IsReadBuffFullPending ( uint32\_t *baseAddr* ) [inline], [static]

The read buffer (or FIFO if the module supports a FIFO) full flag is only cleared by reading it when it is set, then reading the data register by calling the [SPI\\_HAL\\_ReadData\(\)](#). This example code demonstrates how to check the flag, read data, and clear the flag.

```
// Check read buffer flag.
if (SPI_HAL_IsReadBuffFullPending(baseAddr))
{
 // Read the data in the buffer, which also clears the flag.
 byte = SPI_HAL_ReadData(baseAddr);
}
```

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

Current setting of the read buffer full flag.

### 28.2.3.20 static bool SPI\_HAL\_IsTxBuffEmptyPending ( uint32\_t *baseAddr* ) [inline], [static]

To clear the transmit buffer (or FIFO if the module supports a FIFO) empty flag, you must first read the flag when it is set. Then write a new data value into the transmit buffer with a call to the [SPI\\_HAL\\_WriteData\(\)](#). The example code shows how to do this.

```
// Check if transmit buffer is empty.
if (SPI_HAL_IsTxBuffEmptyPending(baseAddr))
{
 // Buffer has room, so write the next data value.
 SPI_HAL_WriteData(baseAddr, byte);
}
```

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

Current setting of the transmit buffer empty flag.

### 28.2.3.21 static bool SPI\_HAL\_IsModeFaultPending ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

Current setting of the mode fault flag.

### 28.2.3.22 void SPI\_HAL\_ClearModeFaultFlag ( uint32\_t *baseAddr* )

## SPI HAL driver

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>baseAddr</i> | Module base address |
|-----------------|---------------------|

**28.2.3.23 static bool SPI\_HAL\_IsMatchPending ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

Current setting of the match flag.

**28.2.3.24 void SPI\_HAL\_ClearMatchFlag ( uint32\_t *baseAddr* )**

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

**28.2.3.25 static uint8\_t SPI\_HAL\_ReadData ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
|-----------------|----------------------|

Returns

The data read from the data buffer.

**28.2.3.26 static void SPI\_HAL\_WriteData ( uint32\_t *baseAddr*, uint8\_t *data* ) [inline], [static]**

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
| <i>data</i>     | The data to send.    |

### 28.2.3.27 void SPI\_HAL\_WriteDataBlocking ( uint32\_t *baseAddr*, uint8\_t *data* )

This function writes data to the SPI data register and waits until the TX is empty to return.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>baseAddr</i> | Module base address. |
| <i>data</i>     | The data to send.    |

### 28.2.3.28 static void SPI\_HAL\_SetMatchValue ( uint32\_t *baseAddr*, uint8\_t *matchByte* ) [inline], [static]

Parameters

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>baseAddr</i>  | Module base address.                          |
| <i>matchByte</i> | The value which triggers the match interrupt. |

### 28.3 SPI Master Peripheral Driver

#### 28.3.1 Overview

This chapter describes the programming interface of the SPI master mode peripheral driver. The SPI master mode peripheral driver transfers data to and from the external devices on the SPI bus in master mode. It provides an easy way to transfer buffers of data with a single function call.

The driver is separated into two implementations: interrupt driven and DMA driven. The interrupt driven driver uses interrupts to alert the CPU that the SPI module needs to service the SPI data transmit and receive operations. The DMA driven driver uses the DMA module to transfer data between the buffers located in memory and the SPI module transmit/receive buffers/FIFOs. Note that some SPI modules may not support DMA transfers and this is distinguished in the driver using the feature name "FSL\_FEATURE\_SPI\_HAS\_DMA\_SUPPORT". The interrupt driven and DMA driven driver APIs are distinguished by the keyword "dma" in the source file name and by the keyword "Dma" in the API name. Each set of drivers have the same API functionality and are described in the following chapters. Note that the DMA driven driver also uses interrupts to alert the CPU that the DMA has completed its transfer or that one final piece of data still needs to be received which is handled by the IRQ handler in the DMA driven driver. In both the interrupt and DMA drivers, the SPI module interrupts are enabled in the NVIC. In addition, the DMA driven driver requests channels from the DMA module. Also, this document will refer to either set of drivers simply as the "SPI master driver" when discussing items that pertain to either driver. Note, when using the DMA driven SPI driver, you will also need to initialize the DMA module. An example is shown later under the Initialization chapter.

The following is a basic step-by-step overview of how to initialize and transfer SPI data. For API specific examples, refer to the examples below. The following uses the interrupt driven APIs and a blocking transfer to illustrate a high-level step-by-step usage. The usage of DMA driver is similar to interrupt driven driver. Keep in mind that using interrupt and DMA drivers in the same runtime application is not normally recommended as you will need to change the SPI interrupt handler. The interrupt driver calls SPI\_DRV\_IRQHandler() and DMA driver calls SPI\_DRV\_DmaIRQHandler(). Refer to files fsl\_spi\_irq.c and fsl\_spi\_dma\_irq.c for an example of these function calls.

```
// Init the SPI
SPI_DRV_MasterInit(masterInstance, &spiMasterState, &userConfig);
// Configure the SPI bus
SPI_DRV_MasterConfigureBus(masterInstance, &spiDevice, &calculatedBaudRate);
// Perform the transfer
SPI_DRV_MasterTransferBlocking(masterInstance, NULL, s_spiSourceBuffer,
 s_spiSinkBuffer, 32, 1000);
// Do other transfers, when done with the SPI, then de-init to shut it down
SPI_DRV_MasterDeinit(instance);
```

Note that it is not normally recommended to mix interrupt and DMA driven drivers in the same application. However, should the user decide to do so, they can separately set up and initialize another instance for DMA operations. The user can also de-init the current interrupt driven SPI instance and re-initialize it for DMA operations. Note that since the DMA driven driver also uses interrupts, the user must take care to direct the IRQ handler from the vector table to the desired driver's IRQ handler. Refer to files fsl\_spi\_irq.c and fsl\_spi\_dma\_irq.c for examples on how to re-direct the IRQ handlers from the vector table to the interrupt-driven and DMA-driven driver IRQ handlers. Such files need to be included in the applications

project in order to direct the SPI interrupt vectors to the proper IRQ handlers. There are also two other files, `fsl_spi_shared_function.c` and `fsl_spi_dma_shared_function.c` that direct the interrupts from the vector table to the appropriate master or slave driver interrupt handler by checking the SPI mode via the HAL function `SPI_HAL_IsMaster(baseAddr)`. Note that the interrupt driver calls `SPI_DRV_IRQHandler()` and DMA driver calls `SPI_DRV_DmaIRQHandler()`. Refer to files `fsl_spi_irq.c` and `fsl_spi_dma_irq.c` for an example of these function calls.

### 28.3.2 SPI Run-time state structures

The SPI master driver uses a run-time state structure to track the ongoing data transfers. The state structure for the interrupt driven driver is called `spi_master_state_t` while the state structure for the DMA driven driver is called `spi_dma_master_state_t`. This structure holds data that the SPI master peripheral driver uses to communicate between the transfer function and the interrupt handler and other driver functions. The interrupt handler in the interrupt driven driver also uses this information to keep track of its progress. The user is only required to pass the memory for the run-time state structure. The SPI master driver populates the members.

### 28.3.3 SPI Device structures

The SPI master driver uses instances of the `spi_device_t` or `spi_dma_device_t` structure to represent the SPI bus configuration required to communicate to an external device that is connected to the bus.

The device structure can be passed into the `SPI_DRV_MasterConfigureBus` or `SPI_DRV_DmaMasterConfigureBus` functions to manually configure the bus for a particular device. For example, if there is only one device connected to the bus, the user might configure it only once. Alternatively the device structure can be passed to the data transfer functions where the bus is reconfigured before the transfer is started. The device structure consists of the following settings: `bitsPerSec` (baud rate in Hz), bit count (if the SPI module supports both 8- and 16-bit transfers), clock polarity and phase, and data shift direction (msb or lsb).

### 28.3.4 SPI Initialization

To initialize the SPI master driver, call the `SPI_DRV_MasterInit()` or `SPI_DRV_DmaMasterInit()` function and pass the instance number of the SPI peripheral you want to use. For example, to use the SPI1 module, pass a value 1 to the initialization function. In addition, the user also passes in the pointer to the run-time state structure used by the master driver to keep track of data transfers.

The user first calls the SPI master initialization to initialize the SPI module, then calls the SPI master configuration bus to configure the module for the specific device on the SPI bus. For the interrupt driven case, while the `SPI_DRV_MasterInit()` function initializes the SPI peripheral, the `SPI_DRV_MasterConfigureBus()` function configures the SPI bus parameters such as bits/frame, clock characteristics, data shift direction, and baud rate. The DMA driven case follows the same logic (except that it uses the DMA API names)

## SPI Master Peripheral Driver

and both examples are provided below. First, the interrupt driven example will be provided followed by the DMA example.

Example code to initialize and configure the SPI master interrupt driven driver including setting up the user configuration and device structures:

```
// Set up and init the master //
uint32_t calculatedBaudRate;
uint32_t masterInstance = 1; // example using SPI instance 1
spi_master_state_t spiMasterState; // simply allocate memory for this

// configure the members of the user config //
spi_master_user_config_t userConfig;
userConfig.polarity = kSpiClockPolarity_ActiveHigh;
userConfig.phase = kSpiClockPhase_FirstEdge;
userConfig.direction = kSpiMsbFirst;
userConfig.bitsPerSec = 5000; // 5KHz baud rate
#if FSL_FEATURE_SPI_16BIT_TRANSFERS
 userConfig.bitCount= kSpi8BitMode; // set only if SPI module supports bit count feature
#endif
// init the SPI module //
SPI_DRV_MasterInit(masterInstance, &spiMasterState);

// configure the SPI bus //
SPI_DRV_MasterConfigureBus(masterInstance, &userConfig, &calculatedBaudRate);
```

Example code to initialize and configure the SPI master DMA driven driver including setting up the user configuration and device structures. Note that some SPI modules may not support DMA transfers and this is distinguished in the driver using the feature name "FSL FEATURE SPI HAS DMA SUPPORT".

```
#if FSL_FEATURE_SPI_HAS_DMA_SUPPORT
 // First, need to init the DMA peripheral driver.
 // NOTE: THIS IS NOT PART OF THE SPI DRIVER. THIS PART INITIALIZES THE DMA DRIVER SO
 // THAT THE SPI DMA DRIVER CAN WORK!
 dma_state_t state; // <- The user simply allocates memory for this structure.
 DMA_DRV_Init(&state);

 // Set up and init the master //
 uint32_t calculatedBaudRate;
 uint32_t masterInstance = 0; // example using SPI instance 0
 spi_dma_master_state_t spiDmaMasterState; // simply allocate memory for this

 // configure the members of the user config //
 spi_dma_master_user_config_t userDmaConfig;
 userDmaConfig.polarity = kSpiClockPolarity_ActiveHigh;
 userDmaConfig.phase = kSpiClockPhase_FirstEdge;
 userDmaConfig.direction = kSpiMsbFirst;
 userDmaConfig.bitsPerSec = 5000; // 5KHz baud rate
#if FSL_FEATURE_SPI_16BIT_TRANSFERS
 userDmaConfig.bitCount= kSpi8BitMode; // set only if SPI module supports bit count feature
#endif
 // init the SPI module //
 SPI_DRV_DmaMasterInit(masterInstance, &spiDmaMasterState);

 // configure the SPI bus //
 SPI_DRV_DmaMasterConfigureBus(masterInstance, &userDmaConfig, &
 calculatedBaudRate);
#endif
```

### 28.3.5 SPI Transfers

The driver supports two different modes to transfer data: blocking and non-blocking. The blocking transfer function waits until the transfer is complete before returning. A timeout parameter is passed into the blocking function. A non-blocking (async) function returns immediately after starting the transfer. It is the responsibility of the user to get the transfer status during the transfer to ascertain when the transfer is complete. As such, additional functions are provided to aid in non-blocking transfers: get transfer status and abort transfer.

Note that some SPI modules may not support DMA transfers.

Blocking transfer function APIs (interrupt and DMA driven):

```
spi_status_t SPI_DRV_MasterTransferBlocking(uint32_t instance,
 const spi_master_user_config_t *
 restrict device,
 const uint8_t * restrict sendBuffer,
 uint8_t * restrict receiveBuffer,
 size_t transferByteCount,
 uint32_t timeout);

spi_status_t SPI_DRV_DmaMasterTransferBlocking(uint32_t
 instance,
 const
 spi_dma_master_user_config_t * restrict device,
 const uint8_t * restrict sendBuffer,
 uint8_t * restrict receiveBuffer,
 size_t transferByteCount,
 uint32_t timeout);
```

Non-blocking function APIs and associated functions (interrupt and DMA driven):

```
// interrupt-driven transfer function
spi_status_t SPI_DRV_MasterTransfer(uint32_t instance,
 const spi_master_user_config_t * restrict device,
 const uint8_t * restrict sendBuffer,
 uint8_t * restrict receiveBuffer,
 size_t transferByteCount);

// Returns whether the previous transfer is completed (interrupt-driven)
spi_status_t SPI_DRV_MasterGetTransferStatus(uint32_t instance,
 uint32_t * bytesTransferred);

// Terminates an asynchronous transfer early (interrupt-driven)
spi_status_t SPI_DRV_MasterAbortTransfer(uint32_t instance);

// DMA-driven transfer function
spi_status_t SPI_DRV_DmaMasterTransfer(uint32_t instance,
 const spi_dma_master_user_config_t *
 restrict device,
 const uint8_t * restrict sendBuffer,
 uint8_t * restrict receiveBuffer,
 size_t transferByteCount);

// Returns whether the previous transfer is completed (DMA-driven)
spi_status_t SPI_DRV_DmaMasterGetTransferStatus(uint32_t
 instance, uint32_t * bytesTransferred);

// Terminates an asynchronous transfer early (DMA-driven)
spi_status_t SPI_DRV_DmaMasterAbortTransfer(uint32_t instance);
```

## SPI Master Peripheral Driver

Example of a blocking transfer (interrupt and DMA driven). Note, first need to initialize the peripheral driver. Refer to the Initialization chapter to perform this first before transferring.

```
// Example blocking transfer function call (interrupt)
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the SPI_DRV_MasterConfigureBus was sufficient, so we'll pass in NULL for the device structure.
// Also, this example shows that we're transferring 32 bytes with a timeout of 1000us.
SPI_DRV_MasterTransferBlocking(masterInstance, NULL, s_spiSourceBuffer,
 s_spiSinkBuffer, 32, 1000);

// Example blocking transfer function call (DMA)
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the SPI_DRV_DmaMasterConfigureBus was sufficient, so we'll pass in NULL for the device
// structure. Also, this example shows that we're transferring 32 bytes with a timeout of 1000us.
SPI_DRV_DmaMasterTransferBlocking(masterInstance, NULL, s_spiSourceBuffer,
 s_spiSinkBuffer,
 32, 1000);
```

Example of a non-blocking transfer (interrupt and DMA driven). Note, first need to initialize the peripheral driver. Refer to the Initialization chapter to perform this first before transferring:

```
// Interrupt Example

uint32_t bytesXfer;

// Example non-blocking transfer function call
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the SPI_DRV_MasterConfigureBus was sufficient, so we'll pass in NULL for the device structure.
// Also, this example shows that we're transferring 32 bytes.
SPI_DRV_MasterTransfer(masterInstance, NULL, s_spiSourceBuffer, s_spiSinkBuffer, 32);

// For non-blocking/async transfers, need to check back to get transfer status, for example
// Where bytesXfer returns the number of bytes transferred
SPI_DRV_MasterGetTransferStatus(masterInstance, &bytesXfer);

// Additionally, if for some reason we need to terminate the on-going transfer:
SPI_DRV_MasterAbortTransfer(masterInstance);

// DMA Example
// Example non-blocking transfer function call
// Note: providing another device structure is optional and for this example, we'll assume the call
// to the SPI_DRV_DmaMasterConfigureBus was sufficient, so we'll pass in NULL for the device
// structure. Also, this example shows that we're transferring 32 bytes.
SPI_DRV_DmaMasterTransfer(masterInstance, NULL, s_spiSourceBuffer, s_spiSinkBuffer
 , 32);

// For non-blocking/async transfers, need to check back to get transfer status, for example
// Where bytesXfer returns the number of bytes transferred
SPI_DRV_DmaMasterGetTransferStatus(masterInstance, &bytesXfer);

// Additionally, if for some reason we need to terminate the on-going transfer:
SPI_DRV_DmaMasterAbortTransfer(masterInstance);
```

### 28.3.6 SPI De-initialization

To de-initialize and shut down the SPI module, call the function:

```
// interrupt driven
```

```
void SPI_DRV_MasterDeinit(masterInstance);

// DMA driven
void SPI_DRV_DmaMasterDeinit(masterInstance);
```

## Data Structures

- struct [spi\\_dma\\_master\\_user\\_config\\_t](#)  
*Information about a device on the SPI bus with DMA.* [More...](#)
- struct [spi\\_dma\\_master\\_state\\_t](#)  
*Runtime state of the SPI master driver with DMA.* [More...](#)
- struct [spi\\_master\\_user\\_config\\_t](#)  
*Information about a device on the SPI bus.* [More...](#)
- struct [spi\\_master\\_state\\_t](#)  
*Runtime state of the SPI master driver.* [More...](#)

## Enumerations

- enum [\\_spi\\_dma\\_timeouts](#) { [kSpiDmaWaitForever](#) = 0x7fffffff }
- enum [\\_spi\\_timeouts](#) { [kSpiWaitForever](#) = 0x7fffffff }

## Variables

- const uint32\_t [g\\_spiBaseAddr](#) []  
*Table of base addresses for SPI instances.*
- const IRQn\_Type [g\\_spiIrqId](#) [HW\_SPI\_INSTANCE\_COUNT]  
*Table to save SPI IRQ enum numbers defined in CMSIS header file.*
- const uint32\_t [g\\_spiBaseAddr](#) []  
*Table of base addresses for SPI instances.*
- const IRQn\_Type [g\\_spiIrqId](#) [HW\_SPI\_INSTANCE\_COUNT]  
*Table to save SPI IRQ enumeration numbers defined in the CMSIS header file.*

## Initialization and shutdown

- void [SPI\\_DRV\\_DmaMasterInit](#) (uint32\_t instance, [spi\\_dma\\_master\\_state\\_t](#) \*spiDmaState)  
*Initializes a SPI instance for master mode operation to work with DMA.*
- void [SPI\\_DRV\\_DmaMasterDeinit](#) (uint32\_t instance)  
*Shuts down a SPI instance with DMA support.*

## Bus configuration

- void [SPI\\_DRV\\_DmaMasterConfigureBus](#) (uint32\_t instance, const [spi\\_dma\\_master\\_user\\_config\\_t](#) \*device, uint32\_t \*calculatedBaudRate)  
*Configures the SPI port to access a device on the bus with DMA support.*

## SPI Master Peripheral Driver

### Blocking transfers

- `spi_status_t SPI_DRV_DmaMasterTransferBlocking (uint32_t instance, const spi_dma_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount, uint32_t timeout)`  
*Performs a blocking SPI master mode transfer with DMA support.*

### Non-blocking transfers

- `spi_status_t SPI_DRV_DmaMasterTransfer (uint32_t instance, const spi_dma_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount)`  
*Performs a non-blocking SPI master mode transfer with DMA support.*
- `spi_status_t SPI_DRV_DmaMasterGetTransferStatus (uint32_t instance, uint32_t *bytesTransferred)`  
*Returns whether the previous transfer finished with DMA support.*
- `spi_status_t SPI_DRV_DmaMasterAbortTransfer (uint32_t instance)`  
*Terminates an asynchronous transfer early with DMA support.*

### Initialization and shutdown

- `void SPI_DRV_MasterInit (uint32_t instance, spi_master_state_t *spiState)`  
*Initializes an SPI instance for master mode operation.*
- `void SPI_DRV_MasterDeinit (uint32_t instance)`  
*Shuts down an SPI instance.*

### Bus configuration

- `void SPI_DRV_MasterConfigureBus (uint32_t instance, const spi_master_user_config_t *device, uint32_t *calculatedBaudRate)`  
*Configures the SPI port to access a device on the bus.*

### Blocking transfers

- `spi_status_t SPI_DRV_MasterTransferBlocking (uint32_t instance, const spi_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount, uint32_t timeout)`  
*Performs a blocking SPI master mode transfer.*

### Non-blocking transfers

- `spi_status_t SPI_DRV_MasterTransfer (uint32_t instance, const spi_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount)`

- **spi\_status\_t SPI\_DRV\_MasterGetTransferStatus** (uint32\_t instance, uint32\_t \*bytesTransferred)
 

*Performs a non-blocking SPI master mode transfer.*

*Returns whether the previous transfer is completed.*
- **spi\_status\_t SPI\_DRV\_MasterAbortTransfer** (uint32\_t instance)
 

*Terminates an asynchronous transfer early.*

## 28.3.7 Data Structure Documentation

### 28.3.7.1 struct spi\_dma\_master\_user\_config\_t

#### Data Fields

- **uint32\_t bitsPerSec**

*SPI baud rate in bits per sec.*

### 28.3.7.2 struct spi\_dma\_master\_state\_t

This structure holds data that are used by the SPI master peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress.

#### Data Fields

- **uint32\_t spiSourceClock**

*Module source clock.*
- **volatile bool isTransferInProgress**

*True if there is an active transfer.*
- **const uint8\_t \*restrict sendBuffer**

*The buffer being sent.*
- **uint8\_t \*restrict receiveBuffer**

*The buffer into which received bytes are placed.*
- **volatile size\_t remainingSendByteCount**

*Number of bytes remaining to send.*
- **volatile size\_t remainingReceiveByteCount**

*Number of bytes remaining to receive.*
- **volatile size\_t transferredByteCount**

*Number of bytes transferred so far.*
- **volatile bool isTransferBlocking**

*True if transfer is a blocking transaction.*
- **semaphore\_t irqSync**

*Used to wait for ISR to complete its business.*
- **bool extraByte**

*Flag used for 16-bit transfers with odd byte count.*
- **dma\_channel\_t dmaReceive**

*The DMA channel used for receive.*
- **dma\_channel\_t dmaTransmit**

*The DMA channel used for transmit.*

## SPI Master Peripheral Driver

- `uint32_t transferByteCnt`  
*Number of bytes to transfer.*

### 28.3.7.2.0.54 Field Documentation

**28.3.7.2.0.54.1 `volatile bool spi_dma_master_state_t::isTransferInProgress`**

**28.3.7.2.0.54.2 `const uint8_t* restrict spi_dma_master_state_t::sendBuffer`**

**28.3.7.2.0.54.3 `uint8_t* restrict spi_dma_master_state_t::receiveBuffer`**

**28.3.7.2.0.54.4 `volatile size_t spi_dma_master_state_t::remainingSendByteCount`**

**28.3.7.2.0.54.5 `volatile size_t spi_dma_master_state_t::remainingReceiveByteCount`**

**28.3.7.2.0.54.6 `volatile size_t spi_dma_master_state_t::transferredByteCount`**

**28.3.7.2.0.54.7 `volatile bool spi_dma_master_state_t::isTransferBlocking`**

**28.3.7.2.0.54.8 `semaphore_t spi_dma_master_state_t::irqSync`**

**28.3.7.2.0.54.9 `uint32_t spi_dma_master_state_t::transferByteCnt`**

### 28.3.7.3 `struct spi_master_user_config_t`

#### Data Fields

- `uint32_t bitsPerSec`  
*SPI baud rate in bits per sec.*

### 28.3.7.4 `struct spi_master_state_t`

This structure holds data that are used by the SPI master peripheral driver to communicate between the transfer function and the interrupt handler. The interrupt handler also uses this information to keep track of its progress.

#### Data Fields

- `uint32_t spiSourceClock`  
*Module source clock.*
- `volatile bool isTransferInProgress`  
*True if there is an active transfer.*
- `const uint8_t *restrict sendBuffer`  
*The buffer being sent.*
- `uint8_t *restrict receiveBuffer`  
*The buffer into which received bytes are placed.*
- `volatile size_t remainingSendByteCount`  
*Number of bytes remaining to send.*
- `volatile size_t remainingReceiveByteCount`

- volatile size\_t **transferredByteCount**  
*Number of bytes remaining to receive.*
- volatile bool **isTransferBlocking**  
*Number of bytes transferred so far.*
- semaphore\_t **irqSync**  
*True if transfer is a blocking transaction.*
- bool **extraByte**  
*Used to wait for ISR to complete its business.*
- bool **extraByte**  
*Flag used for 16-bit transfers with odd byte count.*

#### 28.3.7.4.0.55 Field Documentation

**28.3.7.4.0.55.1 volatile bool spi\_master\_state\_t::isTransferInProgress**

**28.3.7.4.0.55.2 const uint8\_t\* restrict spi\_master\_state\_t::sendBuffer**

**28.3.7.4.0.55.3 uint8\_t\* restrict spi\_master\_state\_t::receiveBuffer**

**28.3.7.4.0.55.4 volatile size\_t spi\_master\_state\_t::remainingSendByteCount**

**28.3.7.4.0.55.5 volatile size\_t spi\_master\_state\_t::remainingReceiveByteCount**

**28.3.7.4.0.55.6 volatile size\_t spi\_master\_state\_t::transferredByteCount**

**28.3.7.4.0.55.7 volatile bool spi\_master\_state\_t::isTransferBlocking**

**28.3.7.4.0.55.8 semaphore\_t spi\_master\_state\_t::irqSync**

#### 28.3.8 Enumeration Type Documentation

##### 28.3.8.1 enum \_spi\_dma\_timeouts

Enumerator

**kSpiDmaWaitForever** Waits forever for a transfer to complete.

##### 28.3.8.2 enum \_spi\_timeouts

Enumerator

**kSpiWaitForever** Waits forever for a transfer to complete.

### 28.3.9 Function Documentation

#### 28.3.9.1 void SPI\_DRV\_DmaMasterInit ( uint32\_t *instance*, spi\_dma\_master\_state\_t \* *spiDmaState* )

This function uses a dma driven method for transferring data. this function initializes the run-time state structure to track the ongoing transfers, ungates the clock to the SPI module, resets the SPI module, initializes the module to user defined settings and default settings, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the SPI module.

This initialization function also configures the DMA module by requesting channels for DMA operation.

Parameters

|                    |                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>    | The instance number of the SPI peripheral.                                                                                                                                                                                                        |
| <i>spiDmaState</i> | The pointer to the SPI DMA master driver state structure. The user must pass the memory for this run-time state structure and the SPI master driver fills out the members. This run-time state structure keeps track of the transfer in progress. |

#### 28.3.9.2 void SPI\_DRV\_DmaMasterDeinit ( uint32\_t *instance* )

This function resets the SPI peripheral, gates its clock, disables any used interrupts to the core, and releases any used DMA channels.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

#### 28.3.9.3 void SPI\_DRV\_DmaMasterConfigureBus ( uint32\_t *instance*, const spi\_dma\_master\_user\_config\_t \* *device*, uint32\_t \* *calculatedBaudRate* )

The term "device" is used to indicate the SPI device for which the SPI master is communicating. The user has two options to configure the device parameters: either pass in the pointer to the device configuration structure to the desired transfer function or pass it in to the SPI\_DRV\_DmaMasterConfigureBus function. The user can pass in a device structure to the transfer function which contains the parameters for the bus (the transfer function then calls this function). However, the user has the option to call this function directly especially to get the calculated baud rate, at which point they may pass in NULL for the device structure in the transfer function (assuming they have called this configure bus function first).

## Parameters

|                            |                                                                                                                                                                                                                                                                                                       |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>            | The instance number of the SPI peripheral.                                                                                                                                                                                                                                                            |
| <i>device</i>              | Pointer to the device information structure. This structure contains the settings for SPI bus configurations.                                                                                                                                                                                         |
| <i>calculated-BaudRate</i> | The calculated baud rate passed back to the user to determine if the calculated baud rate is close enough to meet the needs. The baud rate never exceeds the desired baud rate unless the baud rate requested is less than the absolute minimum in which case the minimum baud rate will be returned. |

**28.3.9.4 `spi_status_t SPI_DRV_DmaMasterTransferBlocking ( uint32_t instance, const spi_dma_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount, uint32_t timeout )`**

This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus. The function does return until the transfer is complete.

## Parameters

|                           |                                                                                                                                                                                                                                          |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>           | The instance number of the SPI peripheral.                                                                                                                                                                                               |
| <i>device</i>             | Pointer to the device information structure. This structure contains the settings for the SPI bus configuration for this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. |
| <i>sendBuffer</i>         | Buffer of data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) are sent.                                                                                                                     |
| <i>receiveBuffer</i>      | Buffer where received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.                                                                                                                             |
| <i>transferByte-Count</i> | The number of bytes to send and receive.                                                                                                                                                                                                 |
| <i>timeout</i>            | A timeout for the transfer in microseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a <a href="#">kStatus_SPI_Timeout</a> error is returned.                                                  |

## SPI Master Peripheral Driver

Returns

#kStatus\_Success The transfer was successful. [kStatus\\_SPI\\_Busy](#) Cannot perform another transfer because one is already in progress. [kStatus\\_SPI\\_Timeout](#) The transfer timed out and was aborted.

### 28.3.9.5 `spi_status_t SPI_DRV_DmaMasterTransfer ( uint32_t instance, const spi_dma_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount )`

This function returns immediately. It is the user's responsibility to check back to ascertain if the transfer is complete (using the `SPI_DRV_DmaMasterGetTransferStatus` function). This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus. The function does return until the transfer is complete.

Parameters

|                           |                                                                                                                                                                                                                                          |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>           | The instance number of the SPI peripheral.                                                                                                                                                                                               |
| <i>device</i>             | Pointer to the device information structure. This structure contains the settings for the SPI bus configuration for this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. |
| <i>sendBuffer</i>         | Buffer of data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) is sent.                                                                                                                      |
| <i>receiveBuffer</i>      | Buffer where received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.                                                                                                                             |
| <i>transferByte-Count</i> | The number of bytes to send and receive.                                                                                                                                                                                                 |

Returns

#kStatus\_Success The transfer was successful. [kStatus\\_SPI\\_Busy](#) Cannot perform another transfer because one is already in progress. [kStatus\\_SPI\\_Timeout](#) The transfer timed out and was aborted.

### 28.3.9.6 `spi_status_t SPI_DRV_DmaMasterGetTransferStatus ( uint32_t instance, uint32_t * bytesTransferred )`

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

|                          |                                                                                                     |
|--------------------------|-----------------------------------------------------------------------------------------------------|
| <i>instance</i>          | The instance number of the SPI peripheral.                                                          |
| <i>bytes-Transferred</i> | Pointer to a value that is filled in with the number of bytes that were sent in the active transfer |

Returns

kStatus\_Success The transfer has completed successfully. kStatus\_SPI\_Busy The transfer is still in progress. *bytesTransferred* is filled with the number of bytes that have been transferred so far.

### 28.3.9.7 **spi\_status\_t SPI\_DRV\_DmaMasterAbortTransfer ( uint32\_t *instance* )**

During an async transfer, the user has the option to terminate the transfer early if the transfer is still in progress.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

Returns

kStatus\_SPI\_Success The transfer was successful. kStatus\_SPI\_NoTransferInProgress No transfer is currently in progress.

### 28.3.9.8 **void SPI\_DRV\_MasterInit ( uint32\_t *instance*, spi\_master\_state\_t \* *spiState* )**

This function uses a CPU interrupt driven method for transferring data. It initializes the run-time state structure to track the ongoing transfers, ungates the clock to the SPI module, resets and initializes the module to default settings, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the SPI module.

Parameters

|                 |                                                                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral.                                                                                                                                                                                                |
| <i>spiState</i> | The pointer to the SPI master driver state structure. The user passes the memory for the run-time state structure and the SPI master driver populates the members. This run-time state structure keeps track of the transfer in progress. |

### 28.3.9.9 **void SPI\_DRV\_MasterDeinit ( uint32\_t *instance* )**

This function resets the SPI peripheral, gates its clock, and disables the interrupt to the core.

## SPI Master Peripheral Driver

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

### 28.3.9.10 void SPI\_DRV\_MasterConfigureBus ( uint32\_t *instance*, const spi\_master\_user\_config\_t \* *device*, uint32\_t \* *calculatedBaudRate* )

The term "device" is used to indicate the SPI device for which the SPI master is communicating. The user has two options to configure the device parameters: either pass in the pointer to the device configuration structure to the desired transfer function (see SPI\_DRV\_MasterTransferDataBlocking or SPI\_DRV\_MasterTransferData) or pass it in to the SPI\_DRV\_MasterConfigureBus function. The user can pass in a device structure to the transfer function which contains the parameters for the bus (the transfer function then calls this function). However, the user has the option to call this function directly especially to get the calculated baud rate, at which point they may pass in NULL for the device structure in the transfer function (assuming they have called this configure bus function first).

Parameters

|                            |                                                                                                                                                                                                                                                                                                       |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>            | The instance number of the SPI peripheral.                                                                                                                                                                                                                                                            |
| <i>device</i>              | Pointer to the device information structure. This structure contains the settings for SPI bus configurations.                                                                                                                                                                                         |
| <i>calculated-BaudRate</i> | The calculated baud rate passed back to the user to determine if the calculated baud rate is close enough to meet the needs. The baud rate never exceeds the desired baud rate unless the baud rate requested is less than the absolute minimum in which case the minimum baud rate will be returned. |

### 28.3.9.11 spi\_status\_t SPI\_DRV\_MasterTransferBlocking ( uint32\_t *instance*, const spi\_master\_user\_config\_t \*restrict *device*, const uint8\_t \*restrict *sendBuffer*, uint8\_t \*restrict *receiveBuffer*, size\_t *transferByteCount*, uint32\_t *timeout* )

This function simultaneously sends and receives data on the SPI bus, because the SPI is a full-duplex bus, and does return until the transfer is complete.

## Parameters

|                          |                                                                                                                                                                                                                                          |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>          | The instance number of the SPI peripheral.                                                                                                                                                                                               |
| <i>device</i>            | Pointer to the device information structure. This structure contains the settings for the SPI bus configuration for this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. |
| <i>sendBuffer</i>        | Buffer of data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) are sent.                                                                                                                     |
| <i>receiveBuffer</i>     | Buffer where received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.                                                                                                                             |
| <i>transferByteCount</i> | The number of bytes to send and receive.                                                                                                                                                                                                 |
| <i>timeout</i>           | A timeout for the transfer in microseconds. If the transfer takes longer than this amount of time, the transfer is aborted and a <a href="#">kStatus_SPI_Timeout</a> error is returned.                                                  |

## Returns

#kStatus\_Success The transfer was successful. [kStatus\\_SPI\\_Busy](#) Cannot perform another transfer because one is already in progress. [kStatus\\_SPI\\_Timeout](#) The transfer timed out and was aborted.

### 28.3.9.12 `spi_status_t SPI_DRV_MasterTransfer ( uint32_t instance, const spi_master_user_config_t *restrict device, const uint8_t *restrict sendBuffer, uint8_t *restrict receiveBuffer, size_t transferByteCount )`

This function returns immediately. The user should check back to find out if the transfer is complete (using the SPI\_DRV\_MasterGetTransferStatus function). This function simultaneously sends and receives data on the SPI bus, because the SPI is a full-duplex bus, and does return until the transfer is complete.

## Parameters

|                 |                                                                                                                                                                                                                                          |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral.                                                                                                                                                                                               |
| <i>device</i>   | Pointer to the device information structure. This structure contains the settings for the SPI bus configuration for this transfer. You may pass NULL for this parameter, in which case the current bus configuration is used unmodified. |

## SPI Master Peripheral Driver

|                           |                                                                                                                     |
|---------------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>sendBuffer</i>         | Buffer of data to send. You may pass NULL for this parameter, in which case bytes with a value of 0 (zero) is sent. |
| <i>receiveBuffer</i>      | Buffer where received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.        |
| <i>transferByte-Count</i> | The number of bytes to send and receive.                                                                            |

Returns

#kStatus\_Success The transfer was successful. [kStatus\\_SPI\\_Busy](#) Cannot perform another transfer because one is already in progress. [kStatus\\_SPI\\_Timeout](#) The transfer timed out and was aborted.

### 28.3.9.13 **spi\_status\_t SPI\_DRV\_MasterGetTransferStatus ( uint32\_t *instance*, uint32\_t \* *bytesTransferred* )**

When performing an a-sync transfer, calling this function shows the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

|                          |                                                                                                     |
|--------------------------|-----------------------------------------------------------------------------------------------------|
| <i>instance</i>          | The instance number of the SPI peripheral.                                                          |
| <i>bytes-Transferred</i> | Pointer to a value that is filled in with the number of bytes that were sent in the active transfer |

kStatus\_Success The transfer has completed successfully. kStatus\_SPI\_Busy The transfer is still in progress. *bytesTransferred* is filled with the number of bytes that have been transferred so far.

### 28.3.9.14 **spi\_status\_t SPI\_DRV\_MasterAbortTransfer ( uint32\_t *instance* )**

During an async transfer, the user has the option to terminate the transfer early if the transfer is still in progress.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

Returns

kStatus\_SPI\_Success The transfer was successful. kStatus\_SPI\_NoTransferInProgress No transfer is currently in progress.

### 28.3.10 Variable Documentation

28.3.10.1 `const uint32_t g_spiBaseAddr[]`

28.3.10.2 `const IRQn_Type g_spilrqId[HW_SPI_INSTANCE_COUNT]`

28.3.10.3 `const uint32_t g_spiBaseAddr[]`

28.3.10.4 `const IRQn_Type g_spilrqId[HW_SPI_INSTANCE_COUNT]`

### 28.4 SPI Slave Peripheral Driver

#### 28.4.1 Overview

This section describes the programming interface of the SPI slave mode peripheral driver.

#### 28.4.2 SPI Overview

The SPI slave peripheral driver provides an easy way to use an SPI peripheral in slave mode. It supports transferring buffers of data with a single function call. When the SPI is configured for slave mode operations, it must first be set up to perform a transfer and then wait for the master to initiate the transfer. The driver is separated into two implementations: interrupt driven and DMA driven. The interrupt driven driver uses interrupts to alert the CPU that the SPI module needs to service the SPI data transmit and receive operations. The DMA driven driver uses the DMA module to transfer data between the buffers located in memory and the SPI module transmit/receive buffers/FIFOs. Note that some SPI modules may not support DMA transfers and this is distinguished in the driver using the feature name "FSL\_FEATURE\_SPI\_HAS\_DMA\_SUPPORT". The interrupt driven and DMA driven driver APIs are distinguished by the keyword "dma" in the source file name and by the keyword "Dma" in the API name. Each set of drivers have the same API functionality and are described in the following chapters. Note that the DMA driven driver also uses interrupts to alert the CPU that the DMA has completed its transfer or that one final piece of data still needs to be received which is handled by the IRQ handler in the DMA driven driver. In both the interrupt and DMA drivers, the SPI module interrupts are enabled in the NVIC. In addition, the DMA driven driver requests channels from the DMA module. Also, this document will refer to either set of drivers simply as the "SPI slave driver" when discussing items that pertain to either driver. Note, when using the DMA driven SPI driver, you will also need to initialize the DMA module. An example is shown later under the Initialization chapter. The following is a basic step-by-step overview of how to setup the SPI for SPI slave mode operations. For API specific examples, refer to the examples below. The following uses the interrupt driven APIs and a blocking transfer to illustrate a high-level step-by-step usage. The usage of DMA driver is similar to interrupt driven driver. Keep in mind that using interrupt and DMA drivers in the same runtime application is not normally recommended as you will need to change the SPI interrupt handler. The interrupt driver calls SPI\_DRV\_IRQHandler() and DMA driver calls SPI\_DRV\_DmaIRQHandler(). Refer to files fsl\_spi\_irq.c and fsl\_spi\_dma\_irq.c for an example of these function calls.

```
// Init the SPI
SPI_DRV_SlaveInit(slaveInstance, &spiSlaveState, &userConfig);
// Perform the transfer (however, waits for master to initiate the transfer)
SPI_DRV_SlaveTransferBlocking(slaveInstance, s_spiSourceBuffer,
 s_spiSinkBuffer, 32, 1000);
// Do other transfers, when done with the SPI, then de-init to shut it down
SPI_DRV_SlaveDeinit(instance);
```

Note that it is not normally recommended to mix interrupt and DMA driven drivers in the same application. However, should the user decide to do so, they can separately set up and initialize another instance for DMA operations. The user can also de-init the current interrupt driven SPI instance and re-initialize it for DMA operations. Note that since the DMA driven driver also uses interrupts, the user must take care to direct

the IRQ handler from the vector table to the desired driver's IRQ handler. Refer to files fsl\_spi\_irq.c and fsl\_spi\_dma\_irq.c for examples on how to re-direct the IRQ handlers from the vector table to the interrupt-driven and DMA-driven driver IRQ handlers. Such files need to be included in the applications project in order to direct the SPI interrupt vectors to the proper IRQ handlers. There are also two other files, fsl\_spi\_shared\_function.c and fsl\_spi\_dma\_shared\_function.c that direct the interrupts from the vector table to the appropriate master or slave driver interrupt handler by checking the SPI mode via the HAL function SPI\_HAL\_IsMaster(baseAddr).

### 28.4.3 SPI Runtime state of the SPI slave driver

The SPI slave driver uses a run-time state structure to track the ongoing data transfers. The state structure for the interrupt driven driver is called `spi_slave_state_t` while the state structure for the DMA driven driver is called `spi_dma_slave_state_t`. The structure holds data that the SPI slave peripheral driver uses to communicate between the transfer function and the interrupt handler and other driver functions. The interrupt handler also uses this information to keep track of its progress. The user is only responsible to pass the memory for this run-time state structure and the SPI slave driver fills out the members.

### 28.4.4 SPI User configuration structures

The SPI slave driver uses instances of the user configuration structure for the SPI slave driver. The user configuration structure for the interrupt driven driver is called `spi_slave_user_config_t` while the user configuration structure for the DMA driven driver is called `spi_dma_slave_user_config_t`. For this reason, the user can configure the most common settings of the SPI peripheral with a single function call.

### 28.4.5 SPI Setup and Initialization

To initialize the SPI slave driver, first create and fill in a `spi_slave_user_config_t` structure for the interrupt driven driver or `spi_dma_slave_user_config_t` structure for the DMA driven driver. This structure defines the data format settings for the SPI peripheral. The structure is not required after the driver is initialized and can be allocated on the stack. The user also must pass the memory for the run-time state structure. Note that some SPI modules may not support DMA transfers and this is distinguished in the driver using the feature name "FSL FEATURE\_SPI\_HAS\_DMA\_SUPPORT".

This is an example code to initialize and configure the driver for interrupt and DMA operations:

```
// declare which module instance you want to use
uint32_t instance = 1;

// Interrupt driven
spi_slave_state_t spiSlaveState;
// update configs
spi_slave_user_config_t slaveUserConfig;
slaveUserConfig.direction = kSpiMsbFirst;
slaveUserConfig.polarity = kSpiClockPolarity_ActiveHigh;
slaveUserConfig.phase = kSpiClockPhase_FirstEdge;
slaveUserConfig.dummyPattern = SPI_DEFAULT_DUMMY_PATTERN;
```

## SPI Slave Peripheral Driver

```
#if FSL_FEATURE_SPI_16BIT_TRANSFERS
 slaveUserConfig.bitCount = kSpi8BitMode;
#endif

 // init the slave (interrupt driven)
 SPI_DRV_SlaveInit(instance, &spiSlaveState, &slaveUserConfig);

#if FSL_FEATURE_SPI_HAS_DMA_SUPPORT
 // DMA driven
 // First set up the DMA peripheral
 dma_state_t state; // <- The user simply allocates memory for this structure.
 DMA_DRV_Init(&state);

 // DMA driven
 spi_dma_slave_state_t spiDmaSlaveState;
 // update configs
 spi_dma_slave_user_config_t slaveDmaUserConfig;
 slaveDmaUserConfig.direction = kSpiMsbFirst;
 slaveDmaUserConfig.polarity = kSpiClockPolarity_ActiveHigh;
 slaveDmaUserConfig.phase = kSpiClockPhase_FirstEdge;
 slaveDmaUserConfig.dummyPattern = SPI_DEFAULT_DUMMY_PATTERN;
#endif FSL_FEATURE_SPI_16BIT_TRANSFERS
 slaveDmaUserConfig.bitCount = kSpi8BitMode;
#endif

 // init the slave (DMA driven)
 SPI_DRV_DmaSlaveInit(instance, &spiDmaSlaveState, &slaveDmaUserConfig);
#endif
```

### 28.4.6 SPI Blocking and non-blocking

The SPI slave driver has two types of transfer functions, blocking and non-blocking calls. With non-blocking calls, the user starts the transfer and then waits for event flags are set. kSpiTransferDone indicates the transmission and reception are done. With the blocking call, the function only returns after the related process is all done.

Here is an example of blocking and non-blocking call (interrupt driven)

```
spi_status_t result;
// Blocking call example
result = SPI_DRV_SlaveTransferBlocking(instance, // number of SPI peripheral
 sendBuffer, // pointer to transmit buffer, can be NULL
 receiveBuffer, // pointer to receive buffer, can be NULL
 transferSize, // size of receive and receive data
 10000); // Time out after 10000ms
// Check the result to know that transferring sucess or not.

// Non-blocking call example
result = SPI_DRV_SlaveTransfer(instance, // number of SPI peripheral
 sendBuffer, // pointer to transmit buffer, can be NULL
 receiveBuffer, // pointer to receive buffer, can be NULL
 transferSize); // size of receive and receive data

// Wait for transfer done
while(kStatus_SPI_Success != SPI_DRV_SlaveGetTransferStatus(instance, NULL));
// Must check the value of osaStatus to know that transferring success or not.
```

Additionally, some SPI modules support DMA transfers. To use the SPI with DMA, see the following example:

```

spi_status_t result;
// Blocking call example
result = SPI_DRV_DmaSlaveTransferBlocking(instance, // number of SPI
 peripheral
 sendBuffer, // pointer to transmit buffer, can be NULL
 receiveBuffer, // pointer to receive buffer, can be NULL
 transferSize, // size of receive and receive data
 10000); // Time out after 10000ms

// Check the result to know that transferring sucess or not.

// Non-blocking call example
result = SPI_DRV_DmaSlaveTransfer(instance, // number of SPI peripheral
 sendBuffer, // pointer to transmit buffer, can be NULL
 receiveBuffer, // pointer to receive buffer, can be NULL
 transferSize); // size of receive and receive data

// Wait for transfer done
while(kStatus_SPI_Success != SPI_DRV_DmaSlaveGetTransferStatus(instance,
NULL));
// Must check the value of osaStatus to know that transferring success or not.

```

## 28.4.7 SPI De-initialization

To de-initialize and shut down the SPI module, call the function:

```

// interrupt driven
void SPI_DRV_SlaveDeinit(masterInstance);

// DMA driven
void SPI_DRV_DmaSlaveDeinit(masterInstance);

```

## Data Structures

- struct [spi\\_dma\\_slave\\_user\\_config\\_t](#)  
*User configuration structure for the SPI slave driver.* [More...](#)
- struct [spi\\_dma\\_slave\\_state\\_t](#)  
*Runtime state of the SPI slave driver.* [More...](#)
- struct [spi\\_slave\\_user\\_config\\_t](#)  
*User configuration structure for the SPI slave driver.* [More...](#)
- struct [spi\\_slave\\_state\\_t](#)  
*Runtime state of the SPI slave driver.* [More...](#)

## Macros

- #define [SPI\\_DMA\\_DEFAULT\\_DUMMY\\_PATTERN](#) (0x0U)  
*Dummy pattern, that SPI slave will send when transmit data was not configured.*
- #define [SPI\\_DEFAULT\\_DUMMY\\_PATTERN](#) (0x0U)  
*Dummy pattern, that SPI slave will send when transmit data was not configured.*

## SPI Slave Peripheral Driver

### Initialization and shutdown

- `spi_status_t SPI_DRV_DmaSlaveInit` (uint32\_t instance, `spi_dma_slave_state_t` \*spiState, const `spi_dma_slave_user_config_t` \*slaveConfig)  
*Initializes a SPI instance for a slave mode operation, using interrupt mechanism.*
- `void SPI_DRV_DmaSlaveDeinit` (uint32\_t instance)  
*Shuts down a SPI instance - interrupt mechanism.*

### Blocking transfers

- `spi_status_t SPI_DRV_DmaSlaveTransferBlocking` (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint32\_t transferByteCount, uint32\_t timeout)  
*Transfers data on SPI bus using interrupt and blocking call.*

### Non-blocking transfers

- `spi_status_t SPI_DRV_DmaSlaveTransfer` (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint32\_t transferByteCount)  
*Starts transfer data on SPI bus using interrupt and non-blocking call.*
- `spi_status_t SPI_DRV_DmaSlaveAbortTransfer` (uint32\_t instance)  
*Abort transfer that started by non-blocking call transfer function.*
- `spi_status_t SPI_DRV_DmaSlaveGetTransferStatus` (uint32\_t instance, uint32\_t \*framesTransferred)  
*Returns whether the previous transfer is finished.*

### Initialization and shutdown

- `spi_status_t SPI_DRV_SlaveInit` (uint32\_t instance, `spi_slave_state_t` \*spiState, const `spi_slave_user_config_t` \*slaveConfig)  
*Initializes a SPI instance for a slave mode operation, using interrupt mechanism.*
- `void SPI_DRV_SlaveDeinit` (uint32\_t instance)  
*Shuts down a SPI instance - interrupt mechanism.*

### Blocking transfers

- `spi_status_t SPI_DRV_SlaveTransferBlocking` (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint32\_t transferByteCount, uint32\_t timeout)  
*Transfers data on SPI bus using interrupt and blocking call.*

### Non-blocking transfers

- `spi_status_t SPI_DRV_SlaveTransfer` (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint32\_t transferByteCount)

- Starts transfer data on SPI bus using interrupt and non-blocking call.
- **spi\_status\_t SPI\_DRV\_SlaveAbortTransfer (uint32\_t instance)**  
Abort transfer that started by non-blocking call transfer function.
- **spi\_status\_t SPI\_DRV\_SlaveGetTransferStatus (uint32\_t instance, uint32\_t \*framesTransferred)**  
Returns whether the previous transfer is finished.

## 28.4.8 Data Structure Documentation

### 28.4.8.1 struct spi\_dma\_slave\_user\_config\_t

#### Data Fields

- **spi\_clock\_phase\_t phase**  
*Clock phase setting.*
- **spi\_clock\_polarity\_t polarity**  
*Clock polarity setting.*
- **spi\_shift\_direction\_t direction**  
*Either LSB or MSB first.*
- **uint16\_t dummyPattern**  
*Dummy data value.*

#### 28.4.8.1.0.56 Field Documentation

##### 28.4.8.1.0.56.1 spi\_clock\_phase\_t spi\_dma\_slave\_user\_config\_t::phase

##### 28.4.8.1.0.56.2 spi\_clock\_polarity\_t spi\_dma\_slave\_user\_config\_t::polarity

##### 28.4.8.1.0.56.3 spi\_shift\_direction\_t spi\_dma\_slave\_user\_config\_t::direction

### 28.4.8.2 struct spi\_dma\_slave\_state\_t

This structure holds data that is used by the SPI slave peripheral driver to communicate between the transfer function and the interrupt handler. The user needs to pass in the memory for this structure and the driver fills out the members.

#### Data Fields

- **spi\_status\_t status**  
*Current state of slave.*
- **event\_t event**  
*Event to notify waiting task.*
- **uint16\_t errorCount**  
*Driver error count.*
- **uint32\_t dummyPattern**  
*Dummy data will be send when do not have data in transmit buffer.*
- **volatile bool isTransferInProgress**  
*True if there is an active transfer.*
- **const uint8\_t \*restrict sendBuffer**

## SPI Slave Peripheral Driver

- *Pointer to transmit buffer.*  
• `uint8_t *restrict receiveBuffer`  
*Pointer to receive buffer.*
- `volatile int32_t remainingSendByteCount`  
*Number of bytes remaining to send.*
- `volatile int32_t remainingReceiveByteCount`  
*Number of bytes remaining to receive.*
- `volatile int32_t transferredByteCount`  
*Number of bytes transferred so far.*
- `bool isSync`  
*Indicates the function call is sync or async.*
- `bool hasExtraByte`  
*Indicates the reception has extra byte.*
- `dma_channel_t dmaReceive`  
*The DMA channel used for receive.*
- `dma_channel_t dmaTransmit`  
*The DMA channel used for transmit.*

### 28.4.8.2.0.57 Field Documentation

#### 28.4.8.2.0.57.1 `volatile bool spi_dma_slave_state_t::isTransferInProgress`

#### 28.4.8.2.0.57.2 `volatile int32_t spi_dma_slave_state_t::remainingSendByteCount`

#### 28.4.8.2.0.57.3 `volatile int32_t spi_dma_slave_state_t::remainingReceiveByteCount`

#### 28.4.8.2.0.57.4 `volatile int32_t spi_dma_slave_state_t::transferredByteCount`

### 28.4.8.3 `struct spi_slave_user_config_t`

#### Data Fields

- `spi_clock_phase_t phase`  
*Clock phase setting.*
- `spi_clock_polarity_t polarity`  
*Clock polarity setting.*
- `spi_shift_direction_t direction`  
*Either LSB or MSB first.*
- `uint16_t dummyPattern`  
*Dummy data value.*

#### 28.4.8.3.0.58 Field Documentation

**28.4.8.3.0.58.1 spi\_clock\_phase\_t spi\_slave\_user\_config\_t::phase**

**28.4.8.3.0.58.2 spi\_clock\_polarity\_t spi\_slave\_user\_config\_t::polarity**

**28.4.8.3.0.58.3 spi\_shift\_direction\_t spi\_slave\_user\_config\_t::direction**

#### 28.4.8.4 struct spi\_slave\_state\_t

This structure holds data that is used by the SPI slave peripheral driver to communicate between the transfer function and the interrupt handler. The user needs to pass in the memory for this structure and the driver fills out the members.

#### Data Fields

- **spi\_status\_t status**  
*Current state of slave.*
- **event\_t event**  
*Event to notify waiting task.*
- **uint16\_t errorCount**  
*Driver error count.*
- **uint32\_t dummyPattern**  
*Dummy data will be send when do not have data in transmit buffer.*
- **volatile bool isTransferInProgress**  
*True if there is an active transfer.*
- **const uint8\_t \*restrict sendBuffer**  
*Pointer to transmit buffer.*
- **uint8\_t \*restrict receiveBuffer**  
*Pointer to receive buffer.*
- **volatile int32\_t remainingSendByteCount**  
*Number of bytes remaining to send.*
- **volatile int32\_t remainingReceiveByteCount**  
*Number of bytes remaining to receive.*
- **volatile int32\_t transferredByteCount**  
*Number of bytes transferred so far.*
- **bool isSync**  
*Indicates the function call is sync or async.*

## SPI Slave Peripheral Driver

### 28.4.8.4.0.59 Field Documentation

28.4.8.4.0.59.1 **volatile bool spi\_slave\_state\_t::isTransferInProgress**

28.4.8.4.0.59.2 **volatile int32\_t spi\_slave\_state\_t::remainingSendByteCount**

28.4.8.4.0.59.3 **volatile int32\_t spi\_slave\_state\_t::remainingReceiveByteCount**

28.4.8.4.0.59.4 **volatile int32\_t spi\_slave\_state\_t::transferredByteCount**

### 28.4.9 Function Documentation

28.4.9.1 **spi\_status\_t SPI\_DRV\_DmaSlaveInit ( uint32\_t *instance*, spi\_dma\_slave\_state\_t \* *spiState*, const spi\_dma\_slave\_user\_config\_t \* *slaveConfig* )**

This function un-gates the clock to the SPI module, initializes the SPI for slave mode. Once initialized, the SPI module is configured in slave mode and user can start transmit, receive data by calls send, receive, transfer functions. This function indicates SPI slave will use interrupt mechanism.

Parameters

|                    |                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------|
| <i>instance</i>    | The instance number of the SPI peripheral.                                                                |
| <i>spiState</i>    | The pointer to the SPI slave driver state structure.                                                      |
| <i>slaveConfig</i> | The configuration structure <a href="#">spi_slave_user_config_t</a> which configures the data bus format. |

Returns

An error code or kStatus\_SPI\_Success.

28.4.9.2 **void SPI\_DRV\_DmaSlaveDeinit ( uint32\_t *instance* )**

Disable SPI module, gates its clock, change SPI slave driver state to NonInit for SPI slave module which is initialized with interrupt mechanism. After de-initialized, user can re-initialize SPI slave module with other mechanisms.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

#### 28.4.9.3 **spi\_status\_t SPI\_DRV\_DmaSlaveTransferBlocking ( uint32\_t *instance*, const uint8\_t \* *sendBuffer*, uint8\_t \* *receiveBuffer*, uint32\_t *transferByteCount*, uint32\_t *timeout* )**

This function check driver status, mechanism and transmit/receive data through SPI bus. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transmit and receive progress will be processed. If only receiveBuffer available, receiving will be processed, else transmitting will be processed. This function only return when its processes were completed. This function uses interrupt mechanism.

Parameters

|                           |                                                                                     |
|---------------------------|-------------------------------------------------------------------------------------|
| <i>instance</i>           | The instance number of SPI peripheral                                               |
| <i>sendBuffer</i>         | The pointer to data that user wants to transmit.                                    |
| <i>receiveBuffer</i>      | The pointer to data that user wants to store received data.                         |
| <i>transferByte-Count</i> | The number of bytes to send and receive.                                            |
| <i>timeout</i>            | The maximum number of miliseconds that function will wait before timed out reached. |

Returns

kStatus\_SPI\_Success if driver starts to send/receive data successfully. kStatus\_SPI\_Error if driver is error and needs to clean error. kStatus\_SPI\_Busy if driver is receiving/transmitting data and not available. kStatus\_SPI\_Timeout if time out reached while transferring is in progress.

#### 28.4.9.4 **spi\_status\_t SPI\_DRV\_DmaSlaveTransfer ( uint32\_t *instance*, const uint8\_t \* *sendBuffer*, uint8\_t \* *receiveBuffer*, uint32\_t *transferByteCount* )**

This function check driver status then set buffer pointers to receive and transmit SPI data. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transfer is done when kDspiTxDone and kDspiRxDone are set. If only receiveBuffer available, transfer is done when kDspiRxDone flag was set, else transfer is done when kDspiTxDone was set. This function uses interrupt mechanism.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | The instance number of SPI peripheral |
|-----------------|---------------------------------------|

## SPI Slave Peripheral Driver

|                           |                                                             |
|---------------------------|-------------------------------------------------------------|
| <i>sendBuffer</i>         | The pointer to data that user wants to transmit.            |
| <i>receiveBuffer</i>      | The pointer to data that user wants to store received data. |
| <i>transferByte-Count</i> | The number of bytes to send and receive.                    |

Returns

kStatus\_SPI\_Success if driver starts to send/receive data successfully. kStatus\_SPI\_Error if driver is error and needs to clean error. kStatus\_SPI\_Busy if driver is receiving/transmitting data and not available.

### 28.4.9.5 spi\_status\_t SPI\_DRV\_DmaSlaveAbortTransfer ( uint32\_t *instance* )

This function stops the transfer which started by function [SPI\\_DRV\\_SlaveTransfer\(\)](#).

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | The instance number of SPI peripheral |
|-----------------|---------------------------------------|

Returns

kStatus\_SPI\_Success if everything is ok. kStatus\_SPI\_InvalidMechanism if the current transaction does not use interrupt mechanism.

### 28.4.9.6 spi\_status\_t SPI\_DRV\_DmaSlaveGetTransferStatus ( uint32\_t *instance*, uint32\_t \* *framesTransferred* )

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

|                           |                                                                                                                                                              |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>           | The instance number of the SPI peripheral.                                                                                                                   |
| <i>frames-Transferred</i> | Pointer to value that is filled in with the number of frames that have been sent in the active transfer. A frame is defined as the number of bits per frame. |

Returns

kStatus\_SPI\_Success The transfer has completed successfully, or kStatus\_SPI\_Busy The transfer is still in progress. framesTransferred is filled with the number of words that have been transferred so far.

#### **28.4.9.7 spi\_status\_t SPI\_DRV\_SlaveInit ( uint32\_t *instance*, spi\_slave\_state\_t \* *spiState*, const spi\_slave\_user\_config\_t \* *slaveConfig* )**

This function un-gates the clock to the SPI module, initializes the SPI for slave mode. Once initialized, the SPI module is configured in slave mode and user can start transmit, receive data by calls send, receive, transfer functions. This function indicates SPI slave will use interrupt mechanism.

Parameters

|                    |                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------------|
| <i>instance</i>    | The instance number of the SPI peripheral.                                                                |
| <i>spiState</i>    | The pointer to the SPI slave driver state structure.                                                      |
| <i>slaveConfig</i> | The configuration structure <a href="#">spi_slave_user_config_t</a> which configures the data bus format. |

Returns

An error code or kStatus\_SPI\_Success.

#### **28.4.9.8 void SPI\_DRV\_SlaveDeinit ( uint32\_t *instance* )**

Disable SPI module, gates its clock, change SPI slave driver state to NonInit for SPI slave module which is initialized with interrupt mechanism. After de-initialized, user can re-initialize SPI slave module with other mechanisms.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

#### **28.4.9.9 spi\_status\_t SPI\_DRV\_SlaveTransferBlocking ( uint32\_t *instance*, const uint8\_t \* *sendBuffer*, uint8\_t \* *receiveBuffer*, uint32\_t *transferByteCount*, uint32\_t *timeout* )**

This function check driver status, mechanism and transmit/receive data through SPI bus. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transmit and receive progress will be processed. If only receiveBuffer available, receiving will be processed, else transmitting will be processed. This function only return when its processes were completed. This function uses interrupt mechanism.

## SPI Slave Peripheral Driver

Parameters

|                          |                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------|
| <i>instance</i>          | The instance number of SPI peripheral                                               |
| <i>sendBuffer</i>        | The pointer to data that user wants to transmit.                                    |
| <i>receiveBuffer</i>     | The pointer to data that user wants to store received data.                         |
| <i>transferByteCount</i> | The number of bytes to send and receive.                                            |
| <i>timeout</i>           | The maximum number of miliseconds that function will wait before timed out reached. |

Returns

kStatus\_SPI\_Success if driver starts to send/receive data successfully. kStatus\_SPI\_Error if driver is error and needs to clean error. kStatus\_SPI\_Busy if driver is receiving/transmitting data and not available. kStatus\_SPI\_Timeout if time out reached while transferring is in progress.

### 28.4.9.10 **spi\_status\_t SPI\_DRV\_SlaveTransfer ( uint32\_t *instance*, const uint8\_t \* *sendBuffer*, uint8\_t \* *receiveBuffer*, uint32\_t *transferByteCount* )**

This function check driver status then set buffer pointers to receive and transmit SPI data. If sendBuffer is NULL, transmit process will be ignored, and if receiveBuffer is NULL receive process is ignored. If both receiveBuffer and sendBuffer available, transfer is done when kDspiTxDone and kDspiRxDone are set. If only receiveBuffer available, transfer is done when kDspiRxDone flag was set, else transfer is done when kDspiTxDone was set. This function uses interrupt mechanism.

Parameters

|                          |                                                             |
|--------------------------|-------------------------------------------------------------|
| <i>instance</i>          | The instance number of SPI peripheral                       |
| <i>sendBuffer</i>        | The pointer to data that user wants to transmit.            |
| <i>receiveBuffer</i>     | The pointer to data that user wants to store received data. |
| <i>transferByteCount</i> | The number of bytes to send and receive.                    |

Returns

kStatus\_SPI\_Success if driver starts to send/receive data successfully. kStatus\_SPI\_Error if driver is error and needs to clean error. kStatus\_SPI\_Busy if driver is receiving/transmitting data and not available.

### 28.4.9.11 **spi\_status\_t SPI\_DRV\_SlaveAbortTransfer ( uint32\_t *instance* )**

This function stops the transfer which started by function [SPI\\_DRV\\_SlaveTransfer\(\)](#).

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | The instance number of SPI peripheral |
|-----------------|---------------------------------------|

Returns

kStatus\_SPI\_Success if everything is ok. kStatus\_SPI\_InvalidMechanism if the current transaction does not use interrupt mechanism.

#### **28.4.9.12 spi\_status\_t SPI\_DRV\_SlaveGetTransferStatus ( uint32\_t *instance*, uint32\_t \* *framesTransferred* )**

When performing an a-sync transfer, the user can call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success). In addition, if the transfer is still in progress, the user can get the number of words that have been transferred up to now.

Parameters

|                           |                                                                                                                                                              |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>           | The instance number of the SPI peripheral.                                                                                                                   |
| <i>frames-Transferred</i> | Pointer to value that is filled in with the number of frames that have been sent in the active transfer. A frame is defined as the number of bits per frame. |

Returns

kStatus\_SPI\_Success The transfer has completed successfully, or kStatus\_SPI\_Busy The transfer is still in progress. *framesTransferred* is filled with the number of words that have been transferred so far.

## Shared SPI Types

### 28.5 Shared SPI Types

This chapter describes SPI driver shared types.

## **28.6 SPI Classes**

This chapter describes the SPI driver C++ classes.



# **Chapter 29**

## **Timer/PWM Module (TPM)**

### **29.1 Overview**

The Kinetis SDK provides both HAL and Peripheral drivers for the Timer/PWM (TPM) Module of Kinetis devices.

### **Modules**

- [TPM HAL driver](#)
- [TPM Peripheral driver](#)

## TPM HAL driver

### 29.2 TPM HAL driver

#### 29.2.1 Overview

The section describes the programming interface of the TPM HAL driver.

## Data Structures

- struct [tpm\\_pwm\\_param\\_t](#)  
*FlexTimer driver PWM parameter. [More...](#)*

## Enumerations

- enum [tpm\\_clock\\_mode\\_t](#)  
*TPM clock source selection for TPM\_SC[CMOD].*
- enum [tpm\\_counting\\_mode\\_t](#)  
*TPM counting mode, up or down.*
- enum [tpm\\_clock\\_ps\\_t](#)  
*TPM prescaler factor selection for clock source.*
- enum [tpm\\_trigger\\_source\\_t](#)  
*TPM trigger sources.*
- enum [tpm\\_pwm\\_mode\\_t](#)  
*TPM operation mode.*
- enum [tpm\\_pwm\\_edge\\_mode\\_t](#)  
*TPM PWM output pulse mode, high-true or low-true on match up.*
- enum [tpm\\_input\\_capture\\_mode\\_t](#)  
*TPM input capture modes.*
- enum [tpm\\_output\\_compare\\_mode\\_t](#)  
*TPM output compare modes.*

## Functions

- static void [TPM\\_HAL\\_SetClockMode](#) (uint32\_t baseAddr, [tpm\\_clock\\_mode\\_t](#) mode)  
*set TPM clock mod.*
- static [tpm\\_clock\\_mode\\_t](#) [TPM\\_HAL\\_GetClockMode](#) (uint32\_t baseAddr)  
*get TPM clock mod.*
- static void [TPM\\_HAL\\_SetClockDiv](#) (uint32\_t baseAddr, [tpm\\_clock\\_ps\\_t](#) ps)  
*set TPM clock divider.*
- static [tpm\\_clock\\_ps\\_t](#) [TPM\\_HAL\\_GetClockDiv](#) (uint32\_t baseAddr)  
*get TPM clock divider.*
- static void [TPM\\_HAL\\_EnableTimerOverflowInt](#) (uint32\_t baseAddr)  
*Enable the TPM peripheral timer overflow interrupt.*
- static void [TPM\\_HAL\\_DisableTimerOverflowInt](#) (uint32\_t baseAddr)  
*Disable the TPM peripheral timer overflow interrupt.*
- static bool [TPM\\_HAL\\_IsOverflowIntEnabled](#) (uint32\_t baseAddr)  
*Read the bit that controls TPM timer overflow interrupt enablement.*
- static bool [TPM\\_HAL\\_GetTimerOverflowStatus](#) (uint32\_t baseAddr)

- static void **TPM\_HAL\_ClearTimerOverflowFlag** (uint32\_t baseAddr)  
*Clear the TPM timer overflow interrupt flag.*
- static void **TPM\_HAL\_SetCpwms** (uint32\_t baseAddr, uint8\_t mode)  
*set TPM center-aligned PWM select.*
- static bool **TPM\_HAL\_GetCpwms** (uint32\_t baseAddr)  
*get TPM center-aligned PWM selection value.*
- static void **TPM\_HAL\_ClearCounter** (uint32\_t baseAddr)  
*clear TPM peripheral current counter value.*
- static uint16\_t **TPM\_HAL\_GetCounterVal** (uint32\_t baseAddr)  
*return TPM peripheral current counter value.*
- static void **TPM\_HAL\_SetMod** (uint32\_t baseAddr, uint16\_t val)  
*set TPM peripheral timer modulo value,*
- static uint16\_t **TPM\_HAL\_GetMod** (uint32\_t baseAddr)  
*return TPM peripheral counter modulo value.*
- static void **TPM\_HAL\_SetChnMsnbaElsnbaVal** (uint32\_t baseAddr, uint8\_t channel, uint8\_t value)  
*Set TPM peripheral timer channel mode and edge level.,*
- static uint8\_t **TPM\_HAL\_GetChnMsnbaVal** (uint32\_t baseAddr, uint8\_t channel)  
*get TPM peripheral timer channel mode,*
- static uint8\_t **TPM\_HAL\_GetChnElsnbaVal** (uint32\_t baseAddr, uint8\_t channel)  
*get TPM peripheral timer channel edge level,*
- static void **TPM\_HAL\_EnableChnInt** (uint32\_t baseAddr, uint8\_t channel)  
*enable TPM peripheral timer channel(n) interrupt.*
- static void **TPM\_HAL\_DisableChnInt** (uint32\_t baseAddr, uint8\_t channel)  
*disable TPM peripheral timer channel(n) interrupt.*
- static bool **TPM\_HAL\_IsChnIntEnabled** (uint32\_t baseAddr, uint8\_t channel)  
*get TPM peripheral timer channel(n) interrupt enabled or not.*
- static bool **TPM\_HAL\_GetChnStatus** (uint32\_t baseAddr, uint8\_t channel)  
*return if any event for TPM peripheral timer channel has occurred ,*
- static void **TPM\_HAL\_ClearChnInt** (uint32\_t baseAddr, uint8\_t channel)  
*return if any event for TPM peripheral timer channel has occurred ,*
- static void **TPM\_HAL\_SetChnCountVal** (uint32\_t baseAddr, uint8\_t channel, uint16\_t val)  
*set TPM peripheral timer channel counter value,*
- static uint16\_t **TPM\_HAL\_GetChnCountVal** (uint32\_t baseAddr, uint8\_t channel)  
*get TPM peripheral timer channel counter value,*
- static uint32\_t **TPM\_HAL\_GetStatusRegVal** (uint32\_t baseAddr)  
*get TPM peripheral timer channel event status*
- static void **TPM\_HAL\_ClearStatusReg** (uint32\_t baseAddr, uint16\_t tpm\_status)  
*clear TPM peripheral timer clear status register value,*
- static void **TPM\_HAL\_SetTriggerSrc** (uint32\_t baseAddr, **tpm\_trigger\_source\_t** trigger\_num)  
*set TPM peripheral timer trigger.*
- static void **TPM\_HAL\_SetTriggerMode** (uint32\_t baseAddr, bool enable)  
*set TPM peripheral timer running on trigger or not .*
- static void **TPM\_HAL\_SetReloadOnTriggerMode** (uint32\_t baseAddr, bool enable)  
*enable TPM timer counter reload on selected trigger or not.*
- static void **TPM\_HAL\_SetStopOnOverflowMode** (uint32\_t baseAddr, bool enable)  
*enable TPM timer counter sotp on selected trigger or not.*
- static void **TPM\_HAL\_EnableGlobalTimeBase** (uint32\_t baseAddr, bool enable)  
*enable TPM timer global time base.*
- static void **TPM\_HAL\_SetDbgMode** (uint32\_t baseAddr, bool enable)  
*set BDM mode.*

## TPM HAL driver

- static void **TPM\_HAL\_SetWaitMode** (uint32\_t baseAddr, bool enable)  
*set WAIT mode behavior.*
- void **TPM\_HAL\_Reset** (uint32\_t baseAddr, uint32\_t instance)  
*reset tpm registers*
- void **TPM\_HAL\_EnablePwmMode** (uint32\_t baseAddr, **tpm\_pwm\_param\_t** \*config, uint8\_t channel)  
*Enables the TPM PWM output mode.*
- void **TPM\_HAL\_DisableChn** (uint32\_t baseAddr, uint8\_t channel)  
*Disables the TPM channel.*

### 29.2.2 Data Structure Documentation

#### 29.2.2.1 struct tpm\_pwm\_param\_t

##### Data Fields

- **tpm\_pwm\_mode\_t mode**  
*TPM PWM operation mode.*
- **tpm\_pwm\_edge\_mode\_t edgeMode**  
*PWM output mode.*
- **uint32\_t uFrequencyHZ**  
*PWM period in Hz.*
- **uint32\_t uDutyCyclePercent**  
*PWM pulse width, value should be between 0 to 100 0=inactive signal(0% duty cycle)...*

##### 29.2.2.1.0.60 Field Documentation

###### 29.2.2.1.0.60.1 uint32\_t tpm\_pwm\_param\_t::uDutyCyclePercent

100=active signal (100% duty cycle).

### 29.2.3 Enumeration Type Documentation

#### 29.2.3.1 enum tpm\_clock\_mode\_t

### 29.2.4 Function Documentation

#### 29.2.4.1 static void TPM\_HAL\_SetClockMode ( **uint32\_t baseAddr**, **tpm\_clock\_mode\_t mode** ) [**inline**], [**static**]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
| <i>mode</i>     | The TPM counter clock    |

#### 29.2.4.2 static tpm\_clock\_mode\_t TPM\_HAL\_GetClockMode ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

#### 29.2.4.3 static void TPM\_HAL\_SetClockDiv ( uint32\_t *baseAddr*, tpm\_clock\_ps\_t *ps* ) [inline], [static]

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | TPM module base address.                  |
| <i>ps</i>       | The TPM peripheral clock prescale divider |

#### 29.2.4.4 static tpm\_clock\_ps\_t TPM\_HAL\_GetClockDiv ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

#### 29.2.4.5 static void TPM\_HAL\_EnableTimerOverflowInt ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

#### 29.2.4.6 static void TPM\_HAL\_DisableTimerOverflowInt ( uint32\_t *baseAddr* ) [inline], [static]

## TPM HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

**29.2.4.7 static bool TPM\_HAL\_IsOverflowIntEnabled ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

Return values

|             |                                                |
|-------------|------------------------------------------------|
| <i>true</i> | if overflow interrupt is enabled, false if not |
|-------------|------------------------------------------------|

**29.2.4.8 static bool TPM\_HAL\_GetTimerOverflowStatus ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

Return values

|             |                           |
|-------------|---------------------------|
| <i>true</i> | if overflow, false if not |
|-------------|---------------------------|

**29.2.4.9 static void TPM\_HAL\_ClearTimerOverflowFlag ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

**29.2.4.10 static void TPM\_HAL\_SetCpwms ( uint32\_t *baseAddr*, uint8\_t *mode* ) [inline], [static]**

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>baseAddr</i> | TPM module base address.                   |
| <i>mode</i>     | 1:upcounting mode 0:up_down counting mode. |

#### 29.2.4.11 static bool TPM\_HAL\_GetCpwms ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

#### 29.2.4.12 static void TPM\_HAL\_ClearCounter ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

#### 29.2.4.13 static uint16\_t TPM\_HAL\_GetCounterVal ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

Return values

|                |                         |
|----------------|-------------------------|
| <i>current</i> | TPM timer counter value |
|----------------|-------------------------|

#### 29.2.4.14 static void TPM\_HAL\_SetMod ( uint32\_t *baseAddr*, uint16\_t *val* ) [inline], [static]

Parameters

## TPM HAL driver

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | TPM module base address.                |
| <i>val</i>      | The value to be set to the timer modulo |

**29.2.4.15 static uint16\_t TPM\_HAL\_GetMod ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

Return values

|            |                    |
|------------|--------------------|
| <i>TPM</i> | timer modula value |
|------------|--------------------|

**29.2.4.16 static void TPM\_HAL\_SetChnMsdbaElsnsbaVal ( uint32\_t *baseAddr*, uint8\_t *channel*, uint8\_t *value* ) [inline], [static]**

TPM channel operate mode, MSnBA and ELSnBA shoud be set at the same time.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>baseAddr</i> | The TPM base address                  |
| <i>channel</i>  | The TPM peripheral channel number     |
| <i>value</i>    | The value to set for MSnBA and ELSnBA |

**29.2.4.17 static uint8\_t TPM\_HAL\_GetChnMsdbaVal ( uint32\_t *baseAddr*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>channel</i>  | The TPM peripheral channel number |

Return values

|            |                                             |
|------------|---------------------------------------------|
| <i>The</i> | MSnB:MSnA mode value, will be 00,01, 10, 11 |
|------------|---------------------------------------------|

29.2.4.18 **static uint8\_t TPM\_HAL\_GetChnElsnbaVal ( uint32\_t *baseAddr*, uint8\_t *channel* ) [inline], [static]**

## TPM HAL driver

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>channel</i>  | The TPM peripheral channel number |

Return values

|            |                                               |
|------------|-----------------------------------------------|
| <i>The</i> | ELSnB:ELSnA mode value, will be 00,01, 10, 11 |
|------------|-----------------------------------------------|

**29.2.4.19 static void TPM\_HAL\_EnableChnInt ( uint32\_t *baseAddr*, uint8\_t *channel* )  
[inline], [static]**

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>channel</i>  | The TPM peripheral channel number |

**29.2.4.20 static void TPM\_HAL\_DisableChnInt ( uint32\_t *baseAddr*, uint8\_t *channel* )  
[inline], [static]**

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>channel</i>  | The TPM peripheral channel number |

**29.2.4.21 static bool TPM\_HAL\_IsChnIntEnabled ( uint32\_t *baseAddr*, uint8\_t *channel* )  
[inline], [static]**

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>channel</i>  | The TPM peripheral channel number |

**29.2.4.22 static bool TPM\_HAL\_GetChnStatus ( uint32\_t *baseAddr*, uint8\_t *channel* )  
[inline], [static]**

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>baseAddr</i> | TPM module base address.           |
| <i>channel</i>  | The TPM peripheral channel number. |

Return values

|             |                                    |
|-------------|------------------------------------|
| <i>true</i> | if event occurred, false otherwise |
|-------------|------------------------------------|

#### 29.2.4.23 static void TPM\_HAL\_ClearChnInt ( uint32\_t *baseAddr*, uint8\_t *channel* ) [inline], [static]

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>baseAddr</i> | TPM module base address.           |
| <i>channel</i>  | The TPM peripheral channel number. |

Return values

|             |                                    |
|-------------|------------------------------------|
| <i>true</i> | if event occurred, false otherwise |
|-------------|------------------------------------|

#### 29.2.4.24 static void TPM\_HAL\_SetChnCountVal ( uint32\_t *baseAddr*, uint8\_t *channel*, uint16\_t *val* ) [inline], [static]

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>baseAddr</i> | TPM module base address.           |
| <i>channel</i>  | The TPM peripheral channel number. |
| <i>val</i>      | counter value to be set            |

#### 29.2.4.25 static uint16\_t TPM\_HAL\_GetChnCountVal ( uint32\_t *baseAddr*, uint8\_t *channel* ) [inline], [static]

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>baseAddr</i> | TPM module base address.           |
| <i>channel</i>  | The TPM peripheral channel number. |

## TPM HAL driver

Return values

|            |                                      |
|------------|--------------------------------------|
| <i>val</i> | return current channel counter value |
|------------|--------------------------------------|

**29.2.4.26 static uint32\_t TPM\_HAL\_GetStatusRegVal ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
|-----------------|--------------------------|

Return values

|            |                                           |
|------------|-------------------------------------------|
| <i>val</i> | return current channel event status value |
|------------|-------------------------------------------|

**29.2.4.27 static void TPM\_HAL\_ClearStatusReg ( uint32\_t *baseAddr*, uint16\_t *tpm\_status* ) [inline], [static]**

Parameters

|                   |                                       |
|-------------------|---------------------------------------|
| <i>baseAddr</i>   | TPM module base address.              |
| <i>tpm_status</i> | tpm channel or overflow flag to clear |

**29.2.4.28 static void TPM\_HAL\_SetTriggerSrc ( uint32\_t *baseAddr*, tpm\_trigger\_source\_t *trigger\_num* ) [inline], [static]**

Parameters

|                    |                          |
|--------------------|--------------------------|
| <i>baseAddr</i>    | TPM module base address. |
| <i>trigger_num</i> | 0-15                     |

**29.2.4.29 static void TPM\_HAL\_SetTriggerMode ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>baseAddr</i> | TPM module base address.    |
| <i>enable</i>   | true to enable, 1 to enable |

#### 29.2.4.30 static void TPM\_HAL\_SetReloadOnTriggerMode ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>enable</i>   | true to enable, false to disable. |

#### 29.2.4.31 static void TPM\_HAL\_SetStopOnOverflowMode ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>enable</i>   | true to enable, false to disable. |

#### 29.2.4.32 static void TPM\_HAL\_EnableGlobalTimeBase ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

Parameters

|                 |                                   |
|-----------------|-----------------------------------|
| <i>baseAddr</i> | TPM module base address.          |
| <i>enable</i>   | true to enable, false to disable. |

#### 29.2.4.33 static void TPM\_HAL\_SetDbgMode ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | TPM module base address.        |
| <i>enable</i>   | false pause, true continue work |

## TPM HAL driver

29.2.4.34 **static void TPM\_HAL\_SetWaitMode ( uint32\_t *baseAddr*, bool *enable* )**  
[**inline**], [**static**]

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>baseAddr</i> | TPM module base address.           |
| <i>enable</i>   | 0 continue running, 1 stop running |

#### 29.2.4.35 void TPM\_HAL\_Reset ( uint32\_t *baseAddr*, uint32\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>baseAddr</i> | TPM module base address.            |
| <i>instance</i> | The TPM peripheral instance number. |

#### 29.2.4.36 void TPM\_HAL\_EnablePwmMode ( uint32\_t *baseAddr*, tpm\_pwm\_param\_t \* *config*, uint8\_t *channel* )

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>baseAddr</i> | TPM module base address.    |
| <i>config</i>   | PWM configuration parameter |
| <i>channel</i>  | The TPM channel number.     |

#### 29.2.4.37 void TPM\_HAL\_DisableChn ( uint32\_t *baseAddr*, uint8\_t *channel* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TPM module base address. |
| <i>channel</i>  | The TPM channel number.  |

### 29.3 TPM Peripheral driver

#### 29.3.1 Overview

The section describes the programming interface of the TPM Peripheral driver. The TPM module is a timer that supports input capture, output compare, and generation of PWM signals. The current SDK driver supports all features.

#### 29.3.2 TPM Initialization

1. To initialize the TPM driver, call the [TPM\\_DRV\\_Init\(\)](#) function and pass the instance number of the TPM you want to use. For instance, to use TPM0, pass a value 0 to the initialization function.
2. Pass a user configuration structure [tpm\\_general\\_config\\_t](#), as shown here:

```
// TPM configuration structure for user. User needs to set the relevant ones.
typedef struct TpmGeneralConfig {
 bool isDBGMode;
 bool isGlobalTimeBase;
 bool isTriggerMode;
 bool isStopCountOnOverflow;
 bool isCountReloadOnTrig;
 tpm_trigger_source_t triggerSource;
} tpm_general_config_t;
```

#### 29.3.3 TPM Generate a PWM signal

Call the [TPM\\_DRV\\_PwmStart\(\)](#) function to generate a PWM signal. Use this structure to configure the different parameters related to this PWM signal.

```
typedef struct TpmPwmParam
{
 tpm_pwm_mode_t mode;
 tpm_pwm_edge_mode_t edgeMode;
 uint32_t uFrequencyHZ;
 uint32_t uDutyCyclePercent;
 0=inactive signal (0% duty cycle)...
 100=active signal (100% duty cycle). //
} tpm_pwm_param_t;
```

The mode options are kTpmEdgeAlignedPWM and kTpmCenterAlignedPWM. For edge mode, the options are kTpmHighTrue and kTpmLowTrue. Specify the PWM signal frequency in Hz and the duty cycle percentage (value between 0-100).

#### 29.3.4 TPM Input Capture

Call the [TPM\\_DRV\\_InputCaptureEnable\(\)](#) function to set up the channel for input capture. Use this enumeration to specify the capture mode to be used.

```
typedef enum _tpm_input_capture_mode_t
{
 kTpmRisingEdge = 1,
 kTpmFallingEdge,
 kTpmRiseOrFallEdge
}tpm_input_capture_mode_t;
```

To enable channel interrupts, use the intEnable argument of the function.

Call the [TPM\\_DRV\\_GetChnVal\(\)](#) function to read the captured LPTPM counter value.

### 29.3.5 TPM Output Compare

Call the [TPM\\_DRV\\_OutputCompareEnable\(\)](#) function to set up the channel for output compare. Use this enumeration to specify the compare mode to be used.

```
typedef enum _tpm_output_compare_mode_t
{
 kTpmOutputNone = 0,
 kTpmToggleOutput,
 kTpmClearOutput,
 kTpmSetOutput,
 kTpmHighPulseOutput,
 kTpmLowPulseOutput
}tpm_output_compare_mode_t;
```

To enable channel interrupts, use the intEnable argument of the function. The matchVal argument of the function is written to the CnV register.

### 29.3.6 TPM Interrupt handler

The TPM driver provides an interrupt handler for the counter overflow and channel interrupts. These handlers clear the status bits.

To add more actions to the default handler, add calls to the functions inside the interrupt handlers `TPMx_IRQHandler()` function, where x=0, 1, 2 depending on the TPM instance.

## Data Structures

- struct [tpm\\_general\\_config\\_t](#)

*Internal driver state information grouped by naming. [More...](#)*

### Enumerations

- enum `tpm_clock_source_t` {  
  kTpmClockSourcNone = 0,  
  kTpmClockSourceModuleHighFreq,  
  kTpmClockSourceModuleOSCERCLK,  
  kTpmClockSourceModuleMCGIRCLK,  
  kTpmClockSourceExternalCLKIN0,  
  kTpmClockSourceExternalCLKIN1,  
  kTpmClockSourceReserved }

*TPM clock source selection.*

### Functions

- void `TPM_DRV_Init` (uint8\_t instance, `tpm_general_config_t` \*info)  
*Initializes the TPM driver.*
- void `TPM_DRV_PwmStop` (uint8\_t instance, `tpm_pwm_param_t` \*param, uint8\_t channel)  
*Stops the channel PWM.*
- bool `TPM_DRV_PwmStart` (uint8\_t instance, `tpm_pwm_param_t` \*param, uint8\_t channel)  
*Configures duty cycle, frequency, and starts outputting PWM on a specified channel.*
- void `TPM_DRV_SetTimeOverflowIntCmd` (uint32\_t instance, bool overflowEnable)  
*Enables or disables the timer overflow interrupt.*
- void `TPM_DRV_SetChnIntCmd` (uint32\_t instance, uint8\_t channelNum, bool enable)  
*Enables or disables the channel interrupt.*
- void `TPM_DRV_SetClock` (uint8\_t instance, `tpm_clock_source_t` clock, `tpm_clock_ps_t` clockPs)  
*Sets the TPM clock source.*
- uint32\_t `TPM_DRV_GetClock` (uint8\_t instance)  
*Gets the TPM clock frequency.*
- void `TPM_DRV_CounterStart` (uint8\_t instance, `tpm_counting_mode_t` countMode, uint32\_t countFinalVal, bool enableOverflowInt)  
*Starts the TPM counter.*
- void `TPM_DRV_CounterStop` (uint8\_t instance)  
*Stops the TPM counter.*
- uint32\_t `TPM_DRV_CounterRead` (uint8\_t instance)  
*Reads back the current value of the TPM counter.*
- void `TPM_DRV_InputCaptureEnable` (uint8\_t instance, uint8\_t channel, `tpm_input_capture_mode_t` mode, uint32\_t countFinalVal, bool intEnable)  
*TPM input capture mode setup.*
- uint32\_t `TPM_DRV_GetChnVal` (uint8\_t instance, uint8\_t channel)  
*Reads back the current value of the TPM channel value.*
- void `TPM_DRV_OutputCompareEnable` (uint8\_t instance, uint8\_t channel, `tpm_output_compare_mode_t` mode, uint32\_t countFinalVal, uint32\_t matchVal, bool intEnable)  
*TPM output compare mode setup.*
- void `TPM_DRV_Deinit` (uint8\_t instance)  
*Shuts down the TPM driver.*
- void `TPM_DRV_IRQHandler` (uint8\_t instance)  
*Action to take when an TPM interrupt is triggered.*

## Variables

- const uint32\_t [g\\_tpmbaseAddr](#) []
 

*Table of base addresses for TPM instances.*
- const IRQn\_Type [g\\_tpmIrqId](#) []
 

*Table to save TPM IRQ numbers for TPM instances.*

### 29.3.7 Data Structure Documentation

#### 29.3.7.1 struct tpm\_general\_config\_t

User needs to set the relevant ones.

#### Data Fields

- bool [isDBGMode](#)

*DBGMode behavioral, false to pause, true to continue run in DBG mode.*
- bool [isGlobalTimeBase](#)

*If Global time base enabled, true to enable, false to disable.*
- bool [isTriggerMode](#)

*If Trigger mode enabled, true to enable, false to disable.*
- bool [isStopCountOnOverflow](#)

*True to stop counter after overflow, false to continue running.*
- bool [isCountReloadOnTrig](#)

*True to reload counter on trigger, false means counter is not reloaded.*
- [tpm\\_trigger\\_source\\_t triggerSource](#)

*Trigger source if trigger mode enabled.*

### 29.3.8 Enumeration Type Documentation

#### 29.3.8.1 enum tpm\_clock\_source\_t

Enumerator

**kTpmClockSourcNone** TPM clock source, None.

**kTpmClockSourceModuleHighFreq** TPM clock source, IRC48MHz or FLL/PLL depending on So-C.

**kTpmClockSourceModuleOSCERCLK** TPM clock source, OSCERCLK.

**kTpmClockSourceModuleMCGIRCLK** TPM clock source, MCGIRCLK.

**kTpmClockSourceExternalCLKIN0** TPM clock source, TPM\_CLKIN0.

**kTpmClockSourceExternalCLKIN1** TPM clock source, TPM\_CLKIN1.

**kTpmClockSourceReserved** TPM clock source, Reserved.

## 29.3.9 Function Documentation

29.3.9.1 `void TPM_DRV_Init( uint8_t instance, tpm_general_config_t * info )`

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
| <i>info</i>     | The TPM peripheral instance info    |

### 29.3.9.2 void TPM\_DRV\_PwmStop ( uint8\_t *instance*, tpm\_pwm\_param\_t \* *param*, uint8\_t *channel* )

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>instance</i> | The TPM peripheral instance number.    |
| <i>param</i>    | PWM parameter to configure PWM options |
| <i>channel</i>  | The channel number.                    |

### 29.3.9.3 bool TPM\_DRV\_PwmStart ( uint8\_t *instance*, tpm\_pwm\_param\_t \* *param*, uint8\_t *channel* )

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>instance</i> | The TPM peripheral instance number.    |
| <i>param</i>    | PWM parameter to configure PWM options |
| <i>channel</i>  | The channel number.                    |

Returns

true: if successful, false: failure to generate the PWM signal

### 29.3.9.4 void TPM\_DRV\_SetTimeOverflowIntCmd ( uint32\_t *instance*, bool *overflowEnable* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

## TPM Peripheral driver

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <i>overflowEnable</i> | true: enable the timer overflow interrupt, false: disable |
|-----------------------|-----------------------------------------------------------|

### 29.3.9.5 void TPM\_DRV\_SetChnIntCmd ( uint32\_t *instance*, uint8\_t *channelNum*, bool *enable* )

Parameters

|                   |                                                    |
|-------------------|----------------------------------------------------|
| <i>instance</i>   | The TPM peripheral instance number.                |
| <i>channelNum</i> | The channel number.                                |
| <i>enable</i>     | true: enable the channel interrupt, false: disable |

### 29.3.9.6 void TPM\_DRV\_SetClock ( uint8\_t *instance*, tpm\_clock\_source\_t *clock*, tpm\_clock\_ps\_t *clockPs* )

Parameters

|                 |                                                                              |
|-----------------|------------------------------------------------------------------------------|
| <i>instance</i> | The TPM peripheral instance number.                                          |
| <i>clock</i>    | The TPM peripheral clock selection. Options are listed in tpm_clock_source_t |
| <i>clockPs</i>  | The TPM peripheral clock prescale factor selection listed in tpm_clock_ps_t  |

### 29.3.9.7 uint32\_t TPM\_DRV\_GetClock ( uint8\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

Returns

The function returns the frequency of the TPM clock.

### 29.3.9.8 void TPM\_DRV\_CounterStart ( uint8\_t *instance*, tpm\_counting\_mode\_t *countMode*, uint32\_t *countFinalVal*, bool *enableOverflowInt* )

This function provides access to the TPM counter. The counter can be run in Up-counting and Up-down counting modes.

Parameters

|                          |                                                       |
|--------------------------|-------------------------------------------------------|
| <i>instance</i>          | The TPM peripheral instance number.                   |
| <i>countMode</i>         | The TPM counter mode defined by tpm_counting_mode_t.  |
| <i>countFinalVal</i>     | The final value that is stored in the MOD register.   |
| <i>enableOverflowInt</i> | true: enable timer overflow interrupt; false: disable |

### 29.3.9.9 void TPM\_DRV\_CounterStop ( uint8\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

### 29.3.9.10 uint32\_t TPM\_DRV\_CounterRead ( uint8\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

### 29.3.9.11 void TPM\_DRV\_InputCaptureEnable ( uint8\_t *instance*, uint8\_t *channel*, tpm\_input\_capture\_mode\_t *mode*, uint32\_t *countFinalVal*, bool *intEnable* )

Parameters

|                      |                                                         |
|----------------------|---------------------------------------------------------|
| <i>instance</i>      | The TPM peripheral instance number.                     |
| <i>channel</i>       | The channel number.                                     |
| <i>mode</i>          | The TPM input mode defined by tpm_input_capture_mode_t. |
| <i>countFinalVal</i> | The final value that is stored in the MOD register.     |
| <i>intEnable</i>     | true: enable channel interrupt; false: disable          |

### 29.3.9.12 uint32\_t TPM\_DRV\_GetChnVal ( uint8\_t *instance*, uint8\_t *channel* )

## TPM Peripheral driver

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
| <i>channel</i>  | The channel number.                 |

**29.3.9.13 void TPM\_DRV\_OutputCompareEnable ( uint8\_t *instance*, uint8\_t *channel*, tpm\_output\_compare\_mode\_t *mode*, uint32\_t *countFinalVal*, uint32\_t *matchVal*, bool *intEnable* )**

Parameters

|                      |                                                           |
|----------------------|-----------------------------------------------------------|
| <i>instance</i>      | The TPM peripheral instance number.                       |
| <i>channel</i>       | The channel number.                                       |
| <i>mode</i>          | The TPM output mode defined by tpm_output_compare_mode_t. |
| <i>countFinalVal</i> | The final value that is stored in the MOD register.       |
| <i>matchVal</i>      | The channel compare value stored in the CnV register      |
| <i>intEnable</i>     | true: enable channel interrupt; false: disable            |

**29.3.9.14 void TPM\_DRV\_Deinit ( uint8\_t *instance* )**

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

**29.3.9.15 void TPM\_DRV\_IRQHandler ( uint8\_t *instance* )**

The timer overflow flag is checked and cleared if set.

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the TPM module. |
|-----------------|------------------------------------|

## 29.3.10 Variable Documentation

**29.3.10.1 const uint32\_t g\_tpmbaseAddr[]**

**29.3.10.2 const IRQn\_Type g\_tpmlrqld[]**

# Chapter 30

## Touch Sense Input (TSI)

The Kinetis SDK provides both HAL and Peripheral drivers for Touch Sense Input drivers (TSI) block of Kinetis devices.

### 30.1 Overview

#### Modules

- [TSI HAL driver](#)  
*The part describes the programming interface of the TSI HAL driver.*
- [TSI Peripheral Driver](#)  
*The part describes the programming interface of the TSI Peripheral driver.*

## TSI HAL driver

### 30.2 TSI HAL driver

The chapter describes the programming interface of the TSI HAL driver.

#### 30.2.1 Overview

##### Files

- file `fsl_tsi_hal.h`
- file `fsl_tsi_v2_hal_specific.h`
- file `fsl_tsi_v4_hal_specific.h`

##### Data Structures

- struct `tsi_n_consecutive_scans_limits_t`  
*TSI low power scan intervals limits. [More...](#)*
- struct `tsi_reference_osc_charge_current_limits_t`  
*TSI Reference oscillator charge current select limits. [More...](#)*
- struct `tsi_external_osc_charge_current_limits_t`  
*TSI External oscillator charge current select limits. [More...](#)*
- struct `tsi_active_mode_prescaler_limits_t`  
*TSI active mode prescaler limits. [More...](#)*
- struct `tsi_parameter_limits_t`  
*TSI operation mode limits. [More...](#)*
- struct `tsi_config_t`  
*TSI configuration structure. [More...](#)*

##### Enumerations

- enum `tsi_status_t` {  
  `kStatus_TSI_Busy`,  
  `kStatus_TSI_LowPower`,  
  `kStatus_TSI_Recalibration`,  
  `kStatus_TSI_InvalidChannel`,  
  `kStatus_TSI_InvalidMode`,  
  `kStatus_TSI_Initialized`,  
  `kStatus_TSI_Error` }
- Error codes for the TSI driver.*
- enum `tsi_n_consecutive_scans_t` {

```

kTsiConsecutiveScansNumber_1time = 0,
kTsiConsecutiveScansNumber_2time = 1,
kTsiConsecutiveScansNumber_3time = 2,
kTsiConsecutiveScansNumber_4time = 3,
kTsiConsecutiveScansNumber_5time = 4,
kTsiConsecutiveScansNumber_6time = 5,
kTsiConsecutiveScansNumber_7time = 6,
kTsiConsecutiveScansNumber_8time = 7,
kTsiConsecutiveScansNumber_9time = 8,
kTsiConsecutiveScansNumber_10time = 9,
kTsiConsecutiveScansNumber_11time = 10,
kTsiConsecutiveScansNumber_12time = 11,
kTsiConsecutiveScansNumber_13time = 12,
kTsiConsecutiveScansNumber_14time = 13,
kTsiConsecutiveScansNumber_15time = 14,
kTsiConsecutiveScansNumber_16time = 15,
kTsiConsecutiveScansNumber_17time = 16,
kTsiConsecutiveScansNumber_18time = 17,
kTsiConsecutiveScansNumber_19time = 18,
kTsiConsecutiveScansNumber_20time = 19,
kTsiConsecutiveScansNumber_21time = 20,
kTsiConsecutiveScansNumber_22time = 21,
kTsiConsecutiveScansNumber_23time = 22,
kTsiConsecutiveScansNumber_24time = 23,
kTsiConsecutiveScansNumber_25time = 24,
kTsiConsecutiveScansNumber_26time = 25,
kTsiConsecutiveScansNumber_27time = 26,
kTsiConsecutiveScansNumber_28time = 27,
kTsiConsecutiveScansNumber_29time = 28,
kTsiConsecutiveScansNumber_30time = 29,
kTsiConsecutiveScansNumber_31time = 30,
kTsiConsecutiveScansNumber_32time = 31 }

```

*TSI number of scan intervals for each electrode.*

- enum `tsi_electrode_osc_prescaler_t` {
 

```

kTsiElecOscPrescaler_1div = 0,
kTsiElecOscPrescaler_2div = 1,
kTsiElecOscPrescaler_4div = 2,
kTsiElecOscPrescaler_8div = 3,
kTsiElecOscPrescaler_16div = 4,
kTsiElecOscPrescaler_32div = 5,
kTsiElecOscPrescaler_64div = 6,
kTsiElecOscPrescaler_128div = 7 }

```

*TSI electrode oscillator prescaler.*

- enum `tsi_low_power_interval_t` {

## TSI HAL driver

```
kTsiLowPowerInterval_1ms = 0,
kTsiLowPowerInterval_5ms = 1,
kTsiLowPowerInterval_10ms = 2,
kTsiLowPowerInterval_15ms = 3,
kTsiLowPowerInterval_20ms = 4,
kTsiLowPowerInterval_30ms = 5,
kTsiLowPowerInterval_40ms = 6,
kTsiLowPowerInterval_50ms = 7,
kTsiLowPowerInterval_75ms = 8,
kTsiLowPowerInterval_100ms = 9,
kTsiLowPowerInterval_125ms = 10,
kTsiLowPowerInterval_150ms = 11,
kTsiLowPowerInterval_200ms = 12,
kTsiLowPowerInterval_300ms = 13,
kTsiLowPowerInterval_400ms = 14,
kTsiLowPowerInterval_500ms = 15 }
```

*TSI low power scan intervals.*

- enum `tsi_reference_osc_charge_current_t` {

```

kTsiRefOscChargeCurrent_1uA = 0,
kTsiRefOscChargeCurrent_2uA = 1,
kTsiRefOscChargeCurrent_3uA = 2,
kTsiRefOscChargeCurrent_4uA = 3,
kTsiRefOscChargeCurrent_5uA = 4,
kTsiRefOscChargeCurrent_6uA = 5,
kTsiRefOscChargeCurrent_7uA = 6,
kTsiRefOscChargeCurrent_8uA = 7,
kTsiRefOscChargeCurrent_9uA = 8,
kTsiRefOscChargeCurrent_10uA = 9,
kTsiRefOscChargeCurrent_11uA = 10,
kTsiRefOscChargeCurrent_12uA = 11,
kTsiRefOscChargeCurrent_13uA = 12,
kTsiRefOscChargeCurrent_14uA = 13,
kTsiRefOscChargeCurrent_15uA = 14,
kTsiRefOscChargeCurrent_16uA = 15,
kTsiRefOscChargeCurrent_17uA = 16,
kTsiRefOscChargeCurrent_18uA = 17,
kTsiRefOscChargeCurrent_19uA = 18,
kTsiRefOscChargeCurrent_20uA = 19,
kTsiRefOscChargeCurrent_21uA = 20,
kTsiRefOscChargeCurrent_22uA = 21,
kTsiRefOscChargeCurrent_23uA = 22,
kTsiRefOscChargeCurrent_24uA = 23,
kTsiRefOscChargeCurrent_25uA = 24,
kTsiRefOscChargeCurrent_26uA = 25,
kTsiRefOscChargeCurrent_27uA = 26,
kTsiRefOscChargeCurrent_28uA = 27,
kTsiRefOscChargeCurrent_29uA = 28,
kTsiRefOscChargeCurrent_30uA = 29,
kTsiRefOscChargeCurrent_31uA = 30,
kTsiRefOscChargeCurrent_32uA = 31,
kTsiRefOscChargeCurrent_500nA = 0,
kTsiRefOscChargeCurrent_1uA = 1,
kTsiRefOscChargeCurrent_2uA = 2,
kTsiRefOscChargeCurrent_4uA = 3,
kTsiRefOscChargeCurrent_8uA = 4,
kTsiRefOscChargeCurrent_16uA = 5,
kTsiRefOscChargeCurrent_32uA = 6,
kTsiRefOscChargeCurrent_64uA = 7 }

```

*TSI Reference oscillator charge current select.*

- enum `tsi_external_osc_charge_current_t` {

## TSI HAL driver

```
kTsiExtOscChargeCurrent_1uA = 0,
kTsiExtOscChargeCurrent_2uA = 1,
kTsiExtOscChargeCurrent_3uA = 2,
kTsiExtOscChargeCurrent_4uA = 3,
kTsiExtOscChargeCurrent_5uA = 4,
kTsiExtOscChargeCurrent_6uA = 5,
kTsiExtOscChargeCurrent_7uA = 6,
kTsiExtOscChargeCurrent_8uA = 7,
kTsiExtOscChargeCurrent_9uA = 8,
kTsiExtOscChargeCurrent_10uA = 9,
kTsiExtOscChargeCurrent_11uA = 10,
kTsiExtOscChargeCurrent_12uA = 11,
kTsiExtOscChargeCurrent_13uA = 12,
kTsiExtOscChargeCurrent_14uA = 13,
kTsiExtOscChargeCurrent_15uA = 14,
kTsiExtOscChargeCurrent_16uA = 15,
kTsiExtOscChargeCurrent_17uA = 16,
kTsiExtOscChargeCurrent_18uA = 17,
kTsiExtOscChargeCurrent_19uA = 18,
kTsiExtOscChargeCurrent_20uA = 19,
kTsiExtOscChargeCurrent_21uA = 20,
kTsiExtOscChargeCurrent_22uA = 21,
kTsiExtOscChargeCurrent_23uA = 22,
kTsiExtOscChargeCurrent_24uA = 23,
kTsiExtOscChargeCurrent_25uA = 24,
kTsiExtOscChargeCurrent_26uA = 25,
kTsiExtOscChargeCurrent_27uA = 26,
kTsiExtOscChargeCurrent_28uA = 27,
kTsiExtOscChargeCurrent_29uA = 28,
kTsiExtOscChargeCurrent_30uA = 29,
kTsiExtOscChargeCurrent_31uA = 30,
kTsiExtOscChargeCurrent_32uA = 31,
kTsiExtOscChargeCurrent_500nA = 0,
kTsiExtOscChargeCurrent_1uA = 1,
kTsiExtOscChargeCurrent_2uA = 2,
kTsiExtOscChargeCurrent_4uA = 3,
kTsiExtOscChargeCurrent_8uA = 4,
kTsiExtOscChargeCurrent_16uA = 5,
kTsiExtOscChargeCurrent_32uA = 6,
kTsiExtOscChargeCurrent_64uA = 7 }
```

*TSI External oscillator charge current select.*

- enum `tsi_internal_cap_trim_t` {

```
kTsiIntCapTrim_0_5pF = 0,
kTsiIntCapTrim_0_6pF = 1,
kTsiIntCapTrim_0_7pF = 2,
kTsiIntCapTrim_0_8pF = 3,
kTsiIntCapTrim_0_9pF = 4,
kTsiIntCapTrim_1_0pF = 5,
kTsiIntCapTrim_1_1pF = 6,
kTsiIntCapTrim_1_2pF = 7 }
```

*TSI Internal capacitance trim value.*

- enum `tsi_osc_delta_voltage_t` {
   
kTsiOscDeltaVoltage\_100mV = 0,
 kTsiOscDeltaVoltage\_150mV = 1,
 kTsiOscDeltaVoltage\_200mV = 2,
 kTsiOscDeltaVoltage\_250mV = 3,
 kTsiOscDeltaVoltage\_300mV = 4,
 kTsiOscDeltaVoltage\_400mV = 5,
 kTsiOscDeltaVoltage\_500mV = 6,
 kTsiOscDeltaVoltage\_600mV = 7 }

*TSI Delta voltage applied to analog oscillators.*

- enum `tsi_active_mode_clock_divider_t` {
   
kTsiActiveClkDiv\_1div = 0,
 kTsiActiveClkDiv\_2048div = 1 }
- TSI Active mode clock divider.*
- enum `tsi_active_mode_clock_source_t` {
   
kTsiActiveClkSource\_BusClock = 0,
 kTsiActiveClkSource\_MCGIRCLK = 1,
 kTsiActiveClkSource\_OSCERCLK = 2 }

*TSI Active mode clock source.*

- enum `tsi_active_mode_prescaler_t` {
   
kTsiActiveModePrescaler\_1div = 0,
 kTsiActiveModePrescaler\_2div = 1,
 kTsiActiveModePrescaler\_4div = 2,
 kTsiActiveModePrescaler\_8div = 3,
 kTsiActiveModePrescaler\_16div = 4,
 kTsiActiveModePrescaler\_32div = 5,
 kTsiActiveModePrescaler\_64div = 6,
 kTsiActiveModePrescaler\_128div = 7 }

*TSI active mode prescaler.*

- enum `tsi_analog_mode_select_t` {
   
kTsiAnalogModeSel\_Capacitive = 0,
 kTsiAnalogModeSel\_NoiseNoFreqLim = 4,
 kTsiAnalogModeSel\_NoiseFreqLim = 8 }

*TSI analog mode select.*

- enum `tsi_reference_osc_charge_current_t` {

## TSI HAL driver

```
kTsiRefOscChargeCurrent_1uA = 0,
kTsiRefOscChargeCurrent_2uA = 1,
kTsiRefOscChargeCurrent_3uA = 2,
kTsiRefOscChargeCurrent_4uA = 3,
kTsiRefOscChargeCurrent_5uA = 4,
kTsiRefOscChargeCurrent_6uA = 5,
kTsiRefOscChargeCurrent_7uA = 6,
kTsiRefOscChargeCurrent_8uA = 7,
kTsiRefOscChargeCurrent_9uA = 8,
kTsiRefOscChargeCurrent_10uA = 9,
kTsiRefOscChargeCurrent_11uA = 10,
kTsiRefOscChargeCurrent_12uA = 11,
kTsiRefOscChargeCurrent_13uA = 12,
kTsiRefOscChargeCurrent_14uA = 13,
kTsiRefOscChargeCurrent_15uA = 14,
kTsiRefOscChargeCurrent_16uA = 15,
kTsiRefOscChargeCurrent_17uA = 16,
kTsiRefOscChargeCurrent_18uA = 17,
kTsiRefOscChargeCurrent_19uA = 18,
kTsiRefOscChargeCurrent_20uA = 19,
kTsiRefOscChargeCurrent_21uA = 20,
kTsiRefOscChargeCurrent_22uA = 21,
kTsiRefOscChargeCurrent_23uA = 22,
kTsiRefOscChargeCurrent_24uA = 23,
kTsiRefOscChargeCurrent_25uA = 24,
kTsiRefOscChargeCurrent_26uA = 25,
kTsiRefOscChargeCurrent_27uA = 26,
kTsiRefOscChargeCurrent_28uA = 27,
kTsiRefOscChargeCurrent_29uA = 28,
kTsiRefOscChargeCurrent_30uA = 29,
kTsiRefOscChargeCurrent_31uA = 30,
kTsiRefOscChargeCurrent_32uA = 31,
kTsiRefOscChargeCurrent_500nA = 0,
kTsiRefOscChargeCurrent_1uA = 1,
kTsiRefOscChargeCurrent_2uA = 2,
kTsiRefOscChargeCurrent_4uA = 3,
kTsiRefOscChargeCurrent_8uA = 4,
kTsiRefOscChargeCurrent_16uA = 5,
kTsiRefOscChargeCurrent_32uA = 6,
kTsiRefOscChargeCurrent_64uA = 7 }
```

*TSI Reference oscillator charge and discharge current select.*

- enum `tsi_oscillator_voltage_rails_t` {  
    `kTsiOscVolRails_Dv_103` = 0,  
    `kTsiOscVolRails_Dv_073` = 1,  
    `kTsiOscVolRails_Dv_043` = 2,

```
kTsiOscVolRails_Dv_029 = 3 }
```

*TSI oscillator's voltage rails.*

- enum `tsi_external_osc_charge_current_t` {
   
kTsiExtOscChargeCurrent\_1uA = 0,  
 kTsiExtOscChargeCurrent\_2uA = 1,  
 kTsiExtOscChargeCurrent\_3uA = 2,  
 kTsiExtOscChargeCurrent\_4uA = 3,  
 kTsiExtOscChargeCurrent\_5uA = 4,  
 kTsiExtOscChargeCurrent\_6uA = 5,  
 kTsiExtOscChargeCurrent\_7uA = 6,  
 kTsiExtOscChargeCurrent\_8uA = 7,  
 kTsiExtOscChargeCurrent\_9uA = 8,  
 kTsiExtOscChargeCurrent\_10uA = 9,  
 kTsiExtOscChargeCurrent\_11uA = 10,  
 kTsiExtOscChargeCurrent\_12uA = 11,  
 kTsiExtOscChargeCurrent\_13uA = 12,  
 kTsiExtOscChargeCurrent\_14uA = 13,  
 kTsiExtOscChargeCurrent\_15uA = 14,  
 kTsiExtOscChargeCurrent\_16uA = 15,  
 kTsiExtOscChargeCurrent\_17uA = 16,  
 kTsiExtOscChargeCurrent\_18uA = 17,  
 kTsiExtOscChargeCurrent\_19uA = 18,  
 kTsiExtOscChargeCurrent\_20uA = 19,  
 kTsiExtOscChargeCurrent\_21uA = 20,  
 kTsiExtOscChargeCurrent\_22uA = 21,  
 kTsiExtOscChargeCurrent\_23uA = 22,  
 kTsiExtOscChargeCurrent\_24uA = 23,  
 kTsiExtOscChargeCurrent\_25uA = 24,  
 kTsiExtOscChargeCurrent\_26uA = 25,  
 kTsiExtOscChargeCurrent\_27uA = 26,  
 kTsiExtOscChargeCurrent\_28uA = 27,  
 kTsiExtOscChargeCurrent\_29uA = 28,  
 kTsiExtOscChargeCurrent\_30uA = 29,  
 kTsiExtOscChargeCurrent\_31uA = 30,  
 kTsiExtOscChargeCurrent\_32uA = 31,  
 kTsiExtOscChargeCurrent\_500nA = 0,  
 kTsiExtOscChargeCurrent\_1uA = 1,  
 kTsiExtOscChargeCurrent\_2uA = 2,  
 kTsiExtOscChargeCurrent\_4uA = 3,  
 kTsiExtOscChargeCurrent\_8uA = 4,  
 kTsiExtOscChargeCurrent\_16uA = 5,  
 kTsiExtOscChargeCurrent\_32uA = 6,  
 kTsiExtOscChargeCurrent\_64uA = 7 }

*TSI External oscillator charge and discharge current select.*

- enum `tsi_channel_number_t` {

## TSI HAL driver

```
kTsiChannelNumber_0 = 0,
kTsiChannelNumber_1 = 1,
kTsiChannelNumber_2 = 2,
kTsiChannelNumber_3 = 3,
kTsiChannelNumber_4 = 4,
kTsiChannelNumber_5 = 5,
kTsiChannelNumber_6 = 6,
kTsiChannelNumber_7 = 7,
kTsiChannelNumber_8 = 8,
kTsiChannelNumber_9 = 9,
kTsiChannelNumber_10 = 10,
kTsiChannelNumber_11 = 11,
kTsiChannelNumber_12 = 12,
kTsiChannelNumber_13 = 13,
kTsiChannelNumber_14 = 14,
kTsiChannelNumber_15 = 15 }
```

*TSI channel number.*

## Functions

- void **TSI\_HAL\_Init** (uint32\_t baseAddr)  
*Initialize hardware.*
- void **TSI\_HAL\_SetConfiguration** (uint32\_t baseAddr, **tsi\_config\_t** \*config)  
*Set configuration of hardware.*
- uint32\_t **TSI\_HAL\_Recalibrate** (uint32\_t baseAddr, **tsi\_config\_t** \*config, const uint32\_t electrodes, const **tsi\_parameter\_limits\_t** \*parLimits)  
*Recalibrate TSI hardware.*
- void **TSI\_HAL\_EnableLowPower** (uint32\_t baseAddr)  
*Enable low power for TSI module.*
- void **TSI\_HAL\_DisableLowPower** (uint32\_t baseAddr)  
*Disable low power for TSI module.*
- static void **TSI\_HAL\_EnableOutOfRangeInterrupt** (uint32\_t baseAddr)  
*Enable out of range interrupt.*
- static void **TSI\_HAL\_EnableEndOfScanInterrupt** (uint32\_t baseAddr)  
*Enable end of scan interrupt.*
- static void **TSI\_HAL\_EnableModule** (uint32\_t baseAddr)  
*Enable Touch Sensing Input Module.*
- static void **TSI\_HAL\_DisableModule** (uint32\_t baseAddr)  
*Disable Touch Sensing Input Module.*
- static uint32\_t **TSI\_HAL\_IsModuleEnabled** (uint32\_t baseAddr)  
*Get module flag enable.*
- static void **TSI\_HAL\_EnableStop** (uint32\_t baseAddr)  
*Enable TSI module in stop mode.*
- static void **TSI\_HAL\_DisableStop** (uint32\_t baseAddr)  
*Disable TSI module in stop mode.*
- static void **TSI\_HAL\_EnableSoftwareTriggerScan** (uint32\_t baseAddr)  
*Enable software trigger scan.*
- static uint32\_t **TSI\_HAL\_GetScanTriggerMode** (uint32\_t baseAddr)

- static uint32\_t **TSI\_HAL\_IsScanInProgress** (uint32\_t baseAddr)
  - Get TSI scan trigger mode.*
- static uint32\_t **TSI\_HAL\_GetOutOfRangeFlag** (uint32\_t baseAddr)
  - Get scan in progress flag.*
- static void **TSI\_HAL\_ClearOutOfRangeFlag** (uint32\_t baseAddr)
  - Clear out of range flag.*
- static uint32\_t **TSI\_HAL\_GetEndOfScanFlag** (uint32\_t baseAddr)
  - Get end of scan flag.*
- static void **TSI\_HAL\_ClearEndOfScanFlag** (uint32\_t baseAddr)
  - Clear end of scan flag.*
- static void **TSI\_HAL\_SetPrescaler** (uint32\_t baseAddr, **tsi\_electrode\_osc\_prescaler\_t** prescaler)
  - Set prescaler.*
- static **tsi\_electrode\_osc\_prescaler\_t TSI\_HAL\_GetPrescaler** (uint32\_t baseAddr)
  - Get prescaler.*
- static void **TSI\_HAL\_SetNumberOfScans** (uint32\_t baseAddr, **tsi\_n\_consecutive\_scans\_t** number)
  - Set number of scans (NSCN).*
- static **tsi\_n\_consecutive\_scans\_t TSI\_HAL\_GetNumberOfScans** (uint32\_t baseAddr)
  - Get number of scans (NSCN).*
- static void **TSI\_HAL\_EnablePeriodicalScan** (uint32\_t baseAddr)
  - Enable periodical (hardware) trigger scan.*
- static void **TSI\_HAL\_EnableErrorInterrupt** (uint32\_t baseAddr)
  - Enable error interrupt.*
- static void **TSI\_HAL\_DisableErrorInterrupt** (uint32\_t baseAddr)
  - Disable error interrupt.*
- static void **TSI\_HAL\_EnableInterrupt** (uint32\_t baseAddr)
  - Enable TSI module interrupt.*
- static void **TSI\_HAL\_DisableInterrupt** (uint32\_t baseAddr)
  - Disable TSI interrupt.*
- static uint32\_t **TSI\_HAL\_IsInterruptEnabled** (uint32\_t baseAddr)
  - Get interrupt enable flag.*
- static void **TSI\_HAL\_StartSoftwareTrigger** (uint32\_t baseAddr)
  - Start measurement (trigger the new measurement).*
- static uint32\_t **TSI\_HAL\_IsOverrun** (uint32\_t baseAddr)
  - Get overrun flag.*
- static void **TSI\_HAL\_ClearOverrunFlag** (uint32\_t baseAddr)
  - Clear over run flag.*
- static uint32\_t **TSI\_HAL\_GetExternalElectrodeErrorFlag** (uint32\_t baseAddr)
  - Get external electrode error flag.*
- static void **TSI\_HAL\_ClearExternalElectrodeErrorFlag** (uint32\_t baseAddr)
  - Clear external electrode error flag.*
- static void **TSI\_HAL\_SetLowPowerScanInterval** (uint32\_t baseAddr, **tsi\_low\_power\_interval\_t** interval)
  - Set low power scan interval.*
- static **tsi\_low\_power\_interval\_t TSI\_HAL\_GetLowPowerScanInterval** (uint32\_t baseAddr)
  - Get low power scan interval.*
- static void **TSI\_HAL\_SetLowPowerClock** (uint32\_t baseAddr, uint32\_t clock)
  - Set low power clock.*
- static uint32\_t **TSI\_HAL\_GetLowPowerClock** (uint32\_t baseAddr)
  - Get low power clock.*

## TSI HAL driver

- static void **TSI\_HAL\_SetReferenceChargeCurrent** (uint32\_t baseAddr, **tsi\_reference\_osc\_charge\_current\_t** current)  
*Set the reference oscillator charge current.*
- static **tsi\_reference\_osc\_charge\_current\_t TSI\_HAL\_GetReferenceChargeCurrent** (uint32\_t baseAddr)  
*Get the reference oscillator charge current.*
- static void **TSI\_HAL\_SetElectrodeChargeCurrent** (uint32\_t baseAddr, **tsi\_external\_osc\_charge\_current\_t** current)  
*Set electrode charge current.*
- static **tsi\_external\_osc\_charge\_current\_t TSI\_HAL\_GetElectrodeChargeCurrent** (uint32\_t baseAddr)  
*Get electrode charge current.*
- static void **TSI\_HAL\_SetScanModulo** (uint32\_t baseAddr, uint32\_t modulo)  
*Set scan modulo value.*
- static uint32\_t **TSI\_HAL\_GetScanModulo** (uint32\_t baseAddr)  
*Get scan modulo value.*
- static void **TSI\_HAL\_SetActiveModeSource** (uint32\_t baseAddr, uint32\_t source)  
*Set active mode source.*
- static uint32\_t **TSI\_HAL\_GetActiveModeSource** (uint32\_t baseAddr)  
*Get active mode source.*
- static void **TSI\_HAL\_SetActiveModePrescaler** (uint32\_t baseAddr, **tsi\_active\_mode\_prescaler\_t** prescaler)  
*Set active mode prescaler.*
- static uint32\_t **TSI\_HAL\_GetActiveModePrescaler** (uint32\_t baseAddr)  
*Get active mode prescaler.*
- static void **TSI\_HAL\_SetLowPowerChannel** (uint32\_t baseAddr, uint32\_t channel)  
*Set low power channel.*
- static uint32\_t **TSI\_HAL\_GetLowPowerChannel** (uint32\_t baseAddr)  
*Get low power channel.*
- static void **TSI\_HAL\_EnableChannel** (uint32\_t baseAddr, uint32\_t channel)  
*Enable channel.*
- static void **TSI\_HAL\_EnableChannels** (uint32\_t baseAddr, uint32\_t channelsMask)  
*Enable channels.*
- static void **TSI\_HAL\_DisableChannel** (uint32\_t baseAddr, uint32\_t channel)  
*Disable channel.*
- static void **TSI\_HAL\_DisableChannels** (uint32\_t baseAddr, uint32\_t channelsMask)  
*Disable channels.*
- static uint32\_t **TSI\_HAL\_GetEnabledChannel** (uint32\_t baseAddr, uint32\_t channel)  
*Returns if channel is enabled.*
- static uint32\_t **TSI\_HAL\_GetEnabledChannels** (uint32\_t baseAddr)  
*Returns mask of enabled channels.*
- static uint16\_t **TSI\_HAL\_GetWakeUpChannelCounter** (uint32\_t baseAddr)  
*Set the Wake up channel counter.*
- static uint32\_t **TSI\_HAL\_GetCounter** (uint32\_t baseAddr, uint32\_t channel)  
*Get tsi counter on actual channel.*
- static void **TSI\_HAL\_SetLowThreshold** (uint32\_t baseAddr, uint32\_t low\_threshold)  
*Set low threshold.*
- static void **TSI\_HAL\_SetHighThreshold** (uint32\_t baseAddr, uint32\_t high\_threshold)  
*Set high threshold.*
- static uint32\_t **TSI\_HAL\_GetEnableStop** (uint32\_t baseAddr)  
*Get TSI STOP enable.*

- static void **TSI\_HAL\_SetEnableStop** (uint32\_t baseAddr)  
*Set TSI STOP enable.*
- static void **TSI\_HAL\_SetDisableStop** (uint32\_t baseAddr)  
*Set TSI STOP disable.*
- static void **TSI\_HAL\_EnableHardwareTriggerScan** (uint32\_t baseAddr)  
*Enable periodical (hardware) trigger scan.*
- static void **TSI\_HAL\_CurrentSourcePairSwapped** (uint32\_t baseAddr)  
*The current sources (CURSW) of electrode oscillator and reference oscillator are swapped.*
- static void **TSI\_HAL\_CurrentSourcePairNotSwapped** (uint32\_t baseAddr)  
*The current sources (CURSW) of electrode oscillator and reference oscillator are not swapped.*
- static uint32\_t **TSI\_HAL\_GetCurrentSourcePairSwapped** (uint32\_t baseAddr)  
*Get current source pair swapped status.*
- static void **TSI\_HAL\_SetMeasuredChannelNumber** (uint32\_t baseAddr, uint32\_t channel)  
*Set the measured channel number.*
- static uint32\_t **TSI\_HAL\_GetMeasuredChannelNumber** (uint32\_t baseAddr)  
*Get the measured channel number.*
- static void **TSI\_HAL\_DmaTransferEnable** (uint32\_t baseAddr)  
*DMA transfer enable.*
- static void **TSI\_HAL\_DmaTransferDisable** (uint32\_t baseAddr)  
*DMA transfer disable - do not generate DMA transfer request.*
- static uint32\_t **TSI\_HAL\_IsDmaTransferEnable** (uint32\_t baseAddr)  
*Get DMA transfer enable flag.*
- static uint32\_t **TSI\_HAL\_GetCounter** (uint32\_t baseAddr)  
*Get conversion counter value.*
- static void **TSI\_HAL\_SetMode** (uint32\_t baseAddr, **tsi\_analog\_mode\_select\_t** mode)  
*Set analog mode of the TSI module.*
- static **tsi\_analog\_mode\_select\_t** **TSI\_HAL\_GetMode** (uint32\_t baseAddr)  
*Get analog mode of the TSI module.*
- static void **TSI\_HAL\_SetOscillatorVoltageRails** (uint32\_t baseAddr, **tsi\_oscilator\_voltage\_rails\_t** dvolt)  
*Set the oscillator's volatage rails.*
- static  
**tsi\_oscilator\_voltage\_rails\_t** **TSI\_HAL\_GetOscillatorVoltageRails** (uint32\_t baseAddr)  
*Get the oscillator's volatage rails.*

### 30.2.1.1 TSI HAL Driver

#### Overview

The TSI HAL driver is designed to control the touch sensing input peripheral and provide basic functions to measure the RAW values of electrode capacitance.

### 30.2.2 Data Structure Documentation

#### 30.2.2.1 **struct tsi\_n\_consecutive\_scans\_limits\_t**

These constants define the limits of the tsi number of consecutive scans in a TSI instance.

### Data Fields

- **tsi\_n\_consecutive\_scans\_t upper**  
*upper limit of number of consecutive scan*
- **tsi\_n\_consecutive\_scans\_t lower**  
*lower limit of number of consecutive scan*

### 30.2.2.2 struct tsi\_reference\_osc\_charge\_current\_limits\_t

These constants define the limits of the TSI Reference oscillator charge current select in a TSI instance.

### Data Fields

- **tsi\_reference\_osc\_charge\_current\_t upper**  
*Reference oscillator charge current upper limit.*
- **tsi\_reference\_osc\_charge\_current\_t lower**  
*Reference oscillator charge current lower limit.*

### 30.2.2.3 struct tsi\_external\_osc\_charge\_current\_limits\_t

These constants define the limits of the TSI External oscillator charge current select in a TSI instance.

### Data Fields

- **tsi\_external\_osc\_charge\_current\_t upper**  
*External oscillator charge current upper limit.*
- **tsi\_external\_osc\_charge\_current\_t lower**  
*External oscillator charge current lower limit.*

### 30.2.2.4 struct tsi\_active\_mode\_prescaler\_limits\_t

These constants define the limits of the TSI active mode prescaler in a TSI instance.

### Data Fields

- **tsi\_active\_mode\_prescaler\_t upper**  
*Input clock source prescaler upper limit.*
- **tsi\_active\_mode\_prescaler\_t lower**  
*Input clock source prescaler lower limit.*

### 30.2.2.5 struct tsi\_parameter\_limits\_t

These constants is used to specify the valid range of settings for the recalibration process of TSI parameters

## Data Fields

- `tsi_n_consecutive_scans_limits_t` `consNumberOfScan`  
*number of consecutive scan limits*
- `tsi_reference_osc_charge_current_limits_t` `refOscChargeCurrent`  
*Reference oscillator charge current limits.*
- `tsi_external_osc_charge_current_limits_t` `extOscChargeCurrent`  
*External oscillator charge current limits.*
- `tsi_active_mode_prescaler_limits_t` `activeModePrescaler`  
*Input clock source prescaler limits.*

### 30.2.2.6 `struct tsi_config_t`

This structure contains the settings for the most common TSI configurations including the TSI module charge currents, number of scans, thresholds etc.

## Data Fields

- `tsi_electrode_osc_prescaler_t` `ps`  
*Prescaler.*
- `tsi_external_osc_charge_current_t` `extchrg`  
*Electrode charge current.*
- `tsi_reference_osc_charge_current_t` `refchrg`  
*Reference charge current.*
- `tsi_n_consecutive_scans_t` `nscn`  
*Number of scans.*
- `tsi_analog_mode_select_t` `mode`  
*TSI mode of operation.*
- `tsi_oscillator_voltage_rails_t` `dvolt`  
*Oscillator's voltage rails.*
- `uint16_t` `thresh`  
*High threshold.*
- `uint16_t` `thresl`  
*Low threshold.*

### 30.2.2.6.0.61 Field Documentation

30.2.2.6.0.61.1 `tsi_n_consecutive_scans_t tsi_config_t::nscn`

30.2.2.6.0.61.2 `tsi_analog_mode_select_t tsi_config_t::mode`

30.2.2.6.0.61.3 `tsi_oscilator_voltage_rails_t tsi_config_t::dvolt`

30.2.2.6.0.61.4 `uint16_t tsi_config_t::thresh`

30.2.2.6.0.61.5 `uint16_t tsi_config_t::thresl`

### 30.2.3 Enumeration Type Documentation

#### 30.2.3.1 enum `tsi_status_t`

Enumerator

*kStatus\_TSI\_Busy* TSI still in progress.

*kStatus\_TSI\_LowPower* TSI is in low power mode.

*kStatus\_TSI\_Recalibration* TSI is under recalibration process.

*kStatus\_TSI\_InvalidChannel* Invalid TSI channel.

*kStatus\_TSI\_InvalidMode* Invalid TSI mode.

*kStatus\_TSI\_Initialized* The driver is initialized and ready to measure.

*kStatus\_TSI\_Error* The general driver error.

#### 30.2.3.2 enum `tsi_n_consecutive_scans_t`

These constants define the tsi number of consecutive scans in a TSI instance for each electrode.

Enumerator

*kTsiConsecutiveScansNumber\_1time* once per electrode

*kTsiConsecutiveScansNumber\_2time* twice per electrode

*kTsiConsecutiveScansNumber\_3time* 3 times consecutive scan

*kTsiConsecutiveScansNumber\_4time* 4 times consecutive scan

*kTsiConsecutiveScansNumber\_5time* 5 times consecutive scan

*kTsiConsecutiveScansNumber\_6time* 6 times consecutive scan

*kTsiConsecutiveScansNumber\_7time* 7 times consecutive scan

*kTsiConsecutiveScansNumber\_8time* 8 times consecutive scan

*kTsiConsecutiveScansNumber\_9time* 9 times consecutive scan

*kTsiConsecutiveScansNumber\_10time* 10 times consecutive scan

*kTsiConsecutiveScansNumber\_11time* 11 times consecutive scan

*kTsiConsecutiveScansNumber\_12time* 12 times consecutive scan

*kTsiConsecutiveScansNumber\_13time* 13 times consecutive scan

*kTsiConsecutiveScansNumber\_14time* 14 times consecutive scan

|                                          |                           |
|------------------------------------------|---------------------------|
| <i>kTsiConsecutiveScansNumber_15time</i> | 15 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_16time</i> | 16 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_17time</i> | 17 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_18time</i> | 18 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_19time</i> | 19 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_20time</i> | 20 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_21time</i> | 21 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_22time</i> | 22 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_23time</i> | 23 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_24time</i> | 24 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_25time</i> | 25 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_26time</i> | 26 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_27time</i> | 27 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_28time</i> | 28 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_29time</i> | 29 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_30time</i> | 30 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_31time</i> | 31 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_32time</i> | 32 times consecutive scan |

### 30.2.3.3 enum tsi\_electrode\_osc\_prescaler\_t

These constants define the tsi electrode oscillator prescaler in a TSI instance.

Enumerator

|                                    |                                                |
|------------------------------------|------------------------------------------------|
| <i>kTsiElecOscPrescaler_1div</i>   | Electrode oscillator frequency divided by 1.   |
| <i>kTsiElecOscPrescaler_2div</i>   | Electrode oscillator frequency divided by 2.   |
| <i>kTsiElecOscPrescaler_4div</i>   | Electrode oscillator frequency divided by 4.   |
| <i>kTsiElecOscPrescaler_8div</i>   | Electrode oscillator frequency divided by 8.   |
| <i>kTsiElecOscPrescaler_16div</i>  | Electrode oscillator frequency divided by 16.  |
| <i>kTsiElecOscPrescaler_32div</i>  | Electrode oscillator frequency divided by 32.  |
| <i>kTsiElecOscPrescaler_64div</i>  | Electrode oscillator frequency divided by 64.  |
| <i>kTsiElecOscPrescaler_128div</i> | Electrode oscillator frequency divided by 128. |

### 30.2.3.4 enum tsi\_low\_power\_interval\_t

These constants define the tsi low power scan intervals in a TSI instance.

Enumerator

|                                  |                    |
|----------------------------------|--------------------|
| <i>kTsiLowPowerInterval_1ms</i>  | 1ms scan interval  |
| <i>kTsiLowPowerInterval_5ms</i>  | 5ms scan interval  |
| <i>kTsiLowPowerInterval_10ms</i> | 10ms scan interval |
| <i>kTsiLowPowerInterval_15ms</i> | 15ms scan interval |

|                                   |                     |
|-----------------------------------|---------------------|
| <i>kTsiLowPowerInterval_20ms</i>  | 20ms scan interval  |
| <i>kTsiLowPowerInterval_30ms</i>  | 30ms scan interval  |
| <i>kTsiLowPowerInterval_40ms</i>  | 40ms scan interval  |
| <i>kTsiLowPowerInterval_50ms</i>  | 50ms scan interval  |
| <i>kTsiLowPowerInterval_75ms</i>  | 75ms scan interval  |
| <i>kTsiLowPowerInterval_100ms</i> | 100ms scan interval |
| <i>kTsiLowPowerInterval_125ms</i> | 125ms scan interval |
| <i>kTsiLowPowerInterval_150ms</i> | 150ms scan interval |
| <i>kTsiLowPowerInterval_200ms</i> | 200ms scan interval |
| <i>kTsiLowPowerInterval_300ms</i> | 300ms scan interval |
| <i>kTsiLowPowerInterval_400ms</i> | 400ms scan interval |
| <i>kTsiLowPowerInterval_500ms</i> | 500ms scan interval |

### 30.2.3.5 enum tsi\_reference\_osc\_charge\_current\_t

These constants define the tsi Reference oscillator charge current select in a TSI instance.

Enumerator

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <i>kTsiRefOscChargeCurrent_1uA</i>  | Reference oscillator charge current is 1uA.  |
| <i>kTsiRefOscChargeCurrent_2uA</i>  | Reference oscillator charge current is 2uA.  |
| <i>kTsiRefOscChargeCurrent_3uA</i>  | Reference oscillator charge current is 3uA.  |
| <i>kTsiRefOscChargeCurrent_4uA</i>  | Reference oscillator charge current is 4uA.  |
| <i>kTsiRefOscChargeCurrent_5uA</i>  | Reference oscillator charge current is 5uA.  |
| <i>kTsiRefOscChargeCurrent_6uA</i>  | Reference oscillator charge current is 6uA.  |
| <i>kTsiRefOscChargeCurrent_7uA</i>  | Reference oscillator charge current is 7uA.  |
| <i>kTsiRefOscChargeCurrent_8uA</i>  | Reference oscillator charge current is 8uA.  |
| <i>kTsiRefOscChargeCurrent_9uA</i>  | Reference oscillator charge current is 9uA.  |
| <i>kTsiRefOscChargeCurrent_10uA</i> | Reference oscillator charge current is 10uA. |
| <i>kTsiRefOscChargeCurrent_11uA</i> | Reference oscillator charge current is 11uA. |
| <i>kTsiRefOscChargeCurrent_12uA</i> | Reference oscillator charge current is 12uA. |
| <i>kTsiRefOscChargeCurrent_13uA</i> | Reference oscillator charge current is 13uA. |
| <i>kTsiRefOscChargeCurrent_14uA</i> | Reference oscillator charge current is 14uA. |
| <i>kTsiRefOscChargeCurrent_15uA</i> | Reference oscillator charge current is 15uA. |
| <i>kTsiRefOscChargeCurrent_16uA</i> | Reference oscillator charge current is 16uA. |
| <i>kTsiRefOscChargeCurrent_17uA</i> | Reference oscillator charge current is 17uA. |
| <i>kTsiRefOscChargeCurrent_18uA</i> | Reference oscillator charge current is 18uA. |
| <i>kTsiRefOscChargeCurrent_19uA</i> | Reference oscillator charge current is 19uA. |
| <i>kTsiRefOscChargeCurrent_20uA</i> | Reference oscillator charge current is 20uA. |
| <i>kTsiRefOscChargeCurrent_21uA</i> | Reference oscillator charge current is 21uA. |
| <i>kTsiRefOscChargeCurrent_22uA</i> | Reference oscillator charge current is 22uA. |
| <i>kTsiRefOscChargeCurrent_23uA</i> | Reference oscillator charge current is 23uA. |
| <i>kTsiRefOscChargeCurrent_24uA</i> | Reference oscillator charge current is 24uA. |
| <i>kTsiRefOscChargeCurrent_25uA</i> | Reference oscillator charge current is 25uA. |

|                                      |                                               |
|--------------------------------------|-----------------------------------------------|
| <i>kTsiRefOscChargeCurrent_26uA</i>  | Reference oscillator charge current is 26uA.  |
| <i>kTsiRefOscChargeCurrent_27uA</i>  | Reference oscillator charge current is 27uA.  |
| <i>kTsiRefOscChargeCurrent_28uA</i>  | Reference oscillator charge current is 28uA.  |
| <i>kTsiRefOscChargeCurrent_29uA</i>  | Reference oscillator charge current is 29uA.  |
| <i>kTsiRefOscChargeCurrent_30uA</i>  | Reference oscillator charge current is 30uA.  |
| <i>kTsiRefOscChargeCurrent_31uA</i>  | Reference oscillator charge current is 31uA.  |
| <i>kTsiRefOscChargeCurrent_32uA</i>  | Reference oscillator charge current is 32uA.  |
| <i>kTsiRefOscChargeCurrent_500nA</i> | Reference oscillator charge current is 500nA. |
| <i>kTsiRefOscChargeCurrent_1uA</i>   | Reference oscillator charge current is 1uA.   |
| <i>kTsiRefOscChargeCurrent_2uA</i>   | Reference oscillator charge current is 2uA.   |
| <i>kTsiRefOscChargeCurrent_4uA</i>   | Reference oscillator charge current is 4uA.   |
| <i>kTsiRefOscChargeCurrent_8uA</i>   | Reference oscillator charge current is 8uA.   |
| <i>kTsiRefOscChargeCurrent_16uA</i>  | Reference oscillator charge current is 16uA.  |
| <i>kTsiRefOscChargeCurrent_32uA</i>  | Reference oscillator charge current is 32uA.  |
| <i>kTsiRefOscChargeCurrent_64uA</i>  | Reference oscillator charge current is 64uA.  |

### 30.2.3.6 enum tsi\_external\_osc\_charge\_current\_t

These constants define the tsi External oscillator charge current select in a TSI instance.

Enumerator

|                                     |                                             |
|-------------------------------------|---------------------------------------------|
| <i>kTsiExtOscChargeCurrent_1uA</i>  | External oscillator charge current is 1uA.  |
| <i>kTsiExtOscChargeCurrent_2uA</i>  | External oscillator charge current is 2uA.  |
| <i>kTsiExtOscChargeCurrent_3uA</i>  | External oscillator charge current is 3uA.  |
| <i>kTsiExtOscChargeCurrent_4uA</i>  | External oscillator charge current is 4uA.  |
| <i>kTsiExtOscChargeCurrent_5uA</i>  | External oscillator charge current is 5uA.  |
| <i>kTsiExtOscChargeCurrent_6uA</i>  | External oscillator charge current is 6uA.  |
| <i>kTsiExtOscChargeCurrent_7uA</i>  | External oscillator charge current is 7uA.  |
| <i>kTsiExtOscChargeCurrent_8uA</i>  | External oscillator charge current is 8uA.  |
| <i>kTsiExtOscChargeCurrent_9uA</i>  | External oscillator charge current is 9uA.  |
| <i>kTsiExtOscChargeCurrent_10uA</i> | External oscillator charge current is 10uA. |
| <i>kTsiExtOscChargeCurrent_11uA</i> | External oscillator charge current is 11uA. |
| <i>kTsiExtOscChargeCurrent_12uA</i> | External oscillator charge current is 12uA. |
| <i>kTsiExtOscChargeCurrent_13uA</i> | External oscillator charge current is 13uA. |
| <i>kTsiExtOscChargeCurrent_14uA</i> | External oscillator charge current is 14uA. |
| <i>kTsiExtOscChargeCurrent_15uA</i> | External oscillator charge current is 15uA. |
| <i>kTsiExtOscChargeCurrent_16uA</i> | External oscillator charge current is 16uA. |
| <i>kTsiExtOscChargeCurrent_17uA</i> | External oscillator charge current is 17uA. |
| <i>kTsiExtOscChargeCurrent_18uA</i> | External oscillator charge current is 18uA. |
| <i>kTsiExtOscChargeCurrent_19uA</i> | External oscillator charge current is 19uA. |
| <i>kTsiExtOscChargeCurrent_20uA</i> | External oscillator charge current is 20uA. |
| <i>kTsiExtOscChargeCurrent_21uA</i> | External oscillator charge current is 21uA. |
| <i>kTsiExtOscChargeCurrent_22uA</i> | External oscillator charge current is 22uA. |

## TSI HAL driver

|                                      |                                              |
|--------------------------------------|----------------------------------------------|
| <i>kTsiExtOscChargeCurrent_23uA</i>  | External oscillator charge current is 23uA.  |
| <i>kTsiExtOscChargeCurrent_24uA</i>  | External oscillator charge current is 24uA.  |
| <i>kTsiExtOscChargeCurrent_25uA</i>  | External oscillator charge current is 25uA.  |
| <i>kTsiExtOscChargeCurrent_26uA</i>  | External oscillator charge current is 26uA.  |
| <i>kTsiExtOscChargeCurrent_27uA</i>  | External oscillator charge current is 27uA.  |
| <i>kTsiExtOscChargeCurrent_28uA</i>  | External oscillator charge current is 28uA.  |
| <i>kTsiExtOscChargeCurrent_29uA</i>  | External oscillator charge current is 29uA.  |
| <i>kTsiExtOscChargeCurrent_30uA</i>  | External oscillator charge current is 30uA.  |
| <i>kTsiExtOscChargeCurrent_31uA</i>  | External oscillator charge current is 31uA.  |
| <i>kTsiExtOscChargeCurrent_32uA</i>  | External oscillator charge current is 32uA.  |
| <i>kTsiExtOscChargeCurrent_500nA</i> | External oscillator charge current is 500nA. |
| <i>kTsiExtOscChargeCurrent_1uA</i>   | External oscillator charge current is 1uA.   |
| <i>kTsiExtOscChargeCurrent_2uA</i>   | External oscillator charge current is 2uA.   |
| <i>kTsiExtOscChargeCurrent_4uA</i>   | External oscillator charge current is 4uA.   |
| <i>kTsiExtOscChargeCurrent_8uA</i>   | External oscillator charge current is 8uA.   |
| <i>kTsiExtOscChargeCurrent_16uA</i>  | External oscillator charge current is 16uA.  |
| <i>kTsiExtOscChargeCurrent_32uA</i>  | External oscillator charge current is 32uA.  |
| <i>kTsiExtOscChargeCurrent_64uA</i>  | External oscillator charge current is 64uA.  |

### 30.2.3.7 enum tsi\_internal\_cap\_trim\_t

These constants define the tsi Internal capacitance trim value in a TSI instance.

Enumerator

|                             |                                       |
|-----------------------------|---------------------------------------|
| <i>kTsiIntCapTrim_0_5pF</i> | 0.5 pF internal reference capacitance |
| <i>kTsiIntCapTrim_0_6pF</i> | 0.6 pF internal reference capacitance |
| <i>kTsiIntCapTrim_0_7pF</i> | 0.7 pF internal reference capacitance |
| <i>kTsiIntCapTrim_0_8pF</i> | 0.8 pF internal reference capacitance |
| <i>kTsiIntCapTrim_0_9pF</i> | 0.9 pF internal reference capacitance |
| <i>kTsiIntCapTrim_1_0pF</i> | 1.0 pF internal reference capacitance |
| <i>kTsiIntCapTrim_1_1pF</i> | 1.1 pF internal reference capacitance |
| <i>kTsiIntCapTrim_1_2pF</i> | 1.2 pF internal reference capacitance |

### 30.2.3.8 enum tsi\_osc\_delta\_voltage\_t

These constants define the tsi Delta voltage applied to analog oscillators in a TSI instance.

Enumerator

|                                  |                                 |
|----------------------------------|---------------------------------|
| <i>kTsiOscDeltaVoltage_100mV</i> | 100 mV delta voltage is applied |
| <i>kTsiOscDeltaVoltage_150mV</i> | 150 mV delta voltage is applied |
| <i>kTsiOscDeltaVoltage_200mV</i> | 200 mV delta voltage is applied |
| <i>kTsiOscDeltaVoltage_250mV</i> | 250 mV delta voltage is applied |

- kTsiOscDeltaVoltage\_300mV*** 300 mV delta voltage is applied
- kTsiOscDeltaVoltage\_400mV*** 400 mV delta voltage is applied
- kTsiOscDeltaVoltage\_500mV*** 500 mV delta voltage is applied
- kTsiOscDeltaVoltage\_600mV*** 600 mV delta voltage is applied

### 30.2.3.9 enum tsi\_active\_mode\_clock\_divider\_t

These constants define the active mode clock divider in a TSI instance.

Enumerator

- kTsiActiveClkDiv\_1div*** Active mode clock divider is set to 1.
- kTsiActiveClkDiv\_2048div*** Active mode clock divider is set to 2048.

### 30.2.3.10 enum tsi\_active\_mode\_clock\_source\_t

These constants define the active mode clock source in a TSI instance.

Enumerator

- kTsiActiveClkSource\_BusClock*** Active mode clock source is set to Bus Clock.
- kTsiActiveClkSource\_MCGIRCLK*** Active mode clock source is set to MCG Internal reference clock.
- kTsiActiveClkSource\_OSCERCLK*** Active mode clock source is set to System oscillator output.

### 30.2.3.11 enum tsi\_active\_mode\_prescaler\_t

These constants define the tsi active mode prescaler in a TSI instance.

Enumerator

- kTsiActiveModePrescaler\_1div*** Input clock source divided by 1.
- kTsiActiveModePrescaler\_2div*** Input clock source divided by 2.
- kTsiActiveModePrescaler\_4div*** Input clock source divided by 4.
- kTsiActiveModePrescaler\_8div*** Input clock source divided by 8.
- kTsiActiveModePrescaler\_16div*** Input clock source divided by 16.
- kTsiActiveModePrescaler\_32div*** Input clock source divided by 32.
- kTsiActiveModePrescaler\_64div*** Input clock source divided by 64.
- kTsiActiveModePrescaler\_128div*** Input clock source divided by 128.

### 30.2.3.12 enum tsi\_analog\_mode\_select\_t

Set up TSI analog modes in a TSI instance.

Enumerator

*kTsiAnalogModeSel\_Capacitive* Active TSI capacitive sensing mode.

*kTsiAnalogModeSel\_NoiseNoFreqLim* TSI works in single threshold noise detection mode and the freq. limitation is disabled

*kTsiAnalogModeSel\_NoiseFreqLim* TSI analog works in single threshold noise detection mode and the freq. limitation is enabled

### 30.2.3.13 enum tsi\_reference\_osc\_charge\_current\_t

These constants define the tsi Reference oscillator charge current select in a TSI (REFCHRG) instance.

Enumerator

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <i>kTsiRefOscChargeCurrent_1uA</i>  | Reference oscillator charge current is 1uA.  |
| <i>kTsiRefOscChargeCurrent_2uA</i>  | Reference oscillator charge current is 2uA.  |
| <i>kTsiRefOscChargeCurrent_3uA</i>  | Reference oscillator charge current is 3uA.  |
| <i>kTsiRefOscChargeCurrent_4uA</i>  | Reference oscillator charge current is 4uA.  |
| <i>kTsiRefOscChargeCurrent_5uA</i>  | Reference oscillator charge current is 5uA.  |
| <i>kTsiRefOscChargeCurrent_6uA</i>  | Reference oscillator charge current is 6uA.  |
| <i>kTsiRefOscChargeCurrent_7uA</i>  | Reference oscillator charge current is 7uA.  |
| <i>kTsiRefOscChargeCurrent_8uA</i>  | Reference oscillator charge current is 8uA.  |
| <i>kTsiRefOscChargeCurrent_9uA</i>  | Reference oscillator charge current is 9uA.  |
| <i>kTsiRefOscChargeCurrent_10uA</i> | Reference oscillator charge current is 10uA. |
| <i>kTsiRefOscChargeCurrent_11uA</i> | Reference oscillator charge current is 11uA. |
| <i>kTsiRefOscChargeCurrent_12uA</i> | Reference oscillator charge current is 12uA. |
| <i>kTsiRefOscChargeCurrent_13uA</i> | Reference oscillator charge current is 13uA. |
| <i>kTsiRefOscChargeCurrent_14uA</i> | Reference oscillator charge current is 14uA. |
| <i>kTsiRefOscChargeCurrent_15uA</i> | Reference oscillator charge current is 15uA. |
| <i>kTsiRefOscChargeCurrent_16uA</i> | Reference oscillator charge current is 16uA. |
| <i>kTsiRefOscChargeCurrent_17uA</i> | Reference oscillator charge current is 17uA. |
| <i>kTsiRefOscChargeCurrent_18uA</i> | Reference oscillator charge current is 18uA. |
| <i>kTsiRefOscChargeCurrent_19uA</i> | Reference oscillator charge current is 19uA. |
| <i>kTsiRefOscChargeCurrent_20uA</i> | Reference oscillator charge current is 20uA. |
| <i>kTsiRefOscChargeCurrent_21uA</i> | Reference oscillator charge current is 21uA. |
| <i>kTsiRefOscChargeCurrent_22uA</i> | Reference oscillator charge current is 22uA. |
| <i>kTsiRefOscChargeCurrent_23uA</i> | Reference oscillator charge current is 23uA. |
| <i>kTsiRefOscChargeCurrent_24uA</i> | Reference oscillator charge current is 24uA. |
| <i>kTsiRefOscChargeCurrent_25uA</i> | Reference oscillator charge current is 25uA. |
| <i>kTsiRefOscChargeCurrent_26uA</i> | Reference oscillator charge current is 26uA. |
| <i>kTsiRefOscChargeCurrent_27uA</i> | Reference oscillator charge current is 27uA. |

|                                      |                                               |
|--------------------------------------|-----------------------------------------------|
| <i>kTsiRefOscChargeCurrent_28uA</i>  | Reference oscillator charge current is 28uA.  |
| <i>kTsiRefOscChargeCurrent_29uA</i>  | Reference oscillator charge current is 29uA.  |
| <i>kTsiRefOscChargeCurrent_30uA</i>  | Reference oscillator charge current is 30uA.  |
| <i>kTsiRefOscChargeCurrent_31uA</i>  | Reference oscillator charge current is 31uA.  |
| <i>kTsiRefOscChargeCurrent_32uA</i>  | Reference oscillator charge current is 32uA.  |
| <i>kTsiRefOscChargeCurrent_500nA</i> | Reference oscillator charge current is 500nA. |
| <i>kTsiRefOscChargeCurrent_1uA</i>   | Reference oscillator charge current is 1uA.   |
| <i>kTsiRefOscChargeCurrent_2uA</i>   | Reference oscillator charge current is 2uA.   |
| <i>kTsiRefOscChargeCurrent_4uA</i>   | Reference oscillator charge current is 4uA.   |
| <i>kTsiRefOscChargeCurrent_8uA</i>   | Reference oscillator charge current is 8uA.   |
| <i>kTsiRefOscChargeCurrent_16uA</i>  | Reference oscillator charge current is 16uA.  |
| <i>kTsiRefOscChargeCurrent_32uA</i>  | Reference oscillator charge current is 32uA.  |
| <i>kTsiRefOscChargeCurrent_64uA</i>  | Reference oscillator charge current is 64uA.  |

### 30.2.3.14 enum tsi\_oscilator\_voltage\_rails\_t

These bits indicate the oscillator's voltage rails.

Enumerator

|                               |                                        |
|-------------------------------|----------------------------------------|
| <i>kTsiOscVolRails_Dv_103</i> | DV = 1.03 V; VP = 1.33 V; Vm = 0.30 V. |
| <i>kTsiOscVolRails_Dv_073</i> | DV = 0.73 V; VP = 1.18 V; Vm = 0.45 V. |
| <i>kTsiOscVolRails_Dv_043</i> | DV = 0.43 V; VP = 1.03 V; Vm = 0.60 V. |
| <i>kTsiOscVolRails_Dv_029</i> | DV = 0.29 V; VP = 0.95 V; Vm = 0.67 V. |

### 30.2.3.15 enum tsi\_external\_osc\_charge\_current\_t

These bits indicate the electrode oscillator charge and discharge current value in TSI (EXTCHRG) instance.

Enumerator

|                                     |                                             |
|-------------------------------------|---------------------------------------------|
| <i>kTsiExtOscChargeCurrent_1uA</i>  | External oscillator charge current is 1uA.  |
| <i>kTsiExtOscChargeCurrent_2uA</i>  | External oscillator charge current is 2uA.  |
| <i>kTsiExtOscChargeCurrent_3uA</i>  | External oscillator charge current is 3uA.  |
| <i>kTsiExtOscChargeCurrent_4uA</i>  | External oscillator charge current is 4uA.  |
| <i>kTsiExtOscChargeCurrent_5uA</i>  | External oscillator charge current is 5uA.  |
| <i>kTsiExtOscChargeCurrent_6uA</i>  | External oscillator charge current is 6uA.  |
| <i>kTsiExtOscChargeCurrent_7uA</i>  | External oscillator charge current is 7uA.  |
| <i>kTsiExtOscChargeCurrent_8uA</i>  | External oscillator charge current is 8uA.  |
| <i>kTsiExtOscChargeCurrent_9uA</i>  | External oscillator charge current is 9uA.  |
| <i>kTsiExtOscChargeCurrent_10uA</i> | External oscillator charge current is 10uA. |
| <i>kTsiExtOscChargeCurrent_11uA</i> | External oscillator charge current is 11uA. |
| <i>kTsiExtOscChargeCurrent_12uA</i> | External oscillator charge current is 12uA. |

|                                      |                                              |
|--------------------------------------|----------------------------------------------|
| <i>kTsiExtOscChargeCurrent_13uA</i>  | External oscillator charge current is 13uA.  |
| <i>kTsiExtOscChargeCurrent_14uA</i>  | External oscillator charge current is 14uA.  |
| <i>kTsiExtOscChargeCurrent_15uA</i>  | External oscillator charge current is 15uA.  |
| <i>kTsiExtOscChargeCurrent_16uA</i>  | External oscillator charge current is 16uA.  |
| <i>kTsiExtOscChargeCurrent_17uA</i>  | External oscillator charge current is 17uA.  |
| <i>kTsiExtOscChargeCurrent_18uA</i>  | External oscillator charge current is 18uA.  |
| <i>kTsiExtOscChargeCurrent_19uA</i>  | External oscillator charge current is 19uA.  |
| <i>kTsiExtOscChargeCurrent_20uA</i>  | External oscillator charge current is 20uA.  |
| <i>kTsiExtOscChargeCurrent_21uA</i>  | External oscillator charge current is 21uA.  |
| <i>kTsiExtOscChargeCurrent_22uA</i>  | External oscillator charge current is 22uA.  |
| <i>kTsiExtOscChargeCurrent_23uA</i>  | External oscillator charge current is 23uA.  |
| <i>kTsiExtOscChargeCurrent_24uA</i>  | External oscillator charge current is 24uA.  |
| <i>kTsiExtOscChargeCurrent_25uA</i>  | External oscillator charge current is 25uA.  |
| <i>kTsiExtOscChargeCurrent_26uA</i>  | External oscillator charge current is 26uA.  |
| <i>kTsiExtOscChargeCurrent_27uA</i>  | External oscillator charge current is 27uA.  |
| <i>kTsiExtOscChargeCurrent_28uA</i>  | External oscillator charge current is 28uA.  |
| <i>kTsiExtOscChargeCurrent_29uA</i>  | External oscillator charge current is 29uA.  |
| <i>kTsiExtOscChargeCurrent_30uA</i>  | External oscillator charge current is 30uA.  |
| <i>kTsiExtOscChargeCurrent_31uA</i>  | External oscillator charge current is 31uA.  |
| <i>kTsiExtOscChargeCurrent_32uA</i>  | External oscillator charge current is 32uA.  |
| <i>kTsiExtOscChargeCurrent_500nA</i> | External oscillator charge current is 500nA. |
| <i>kTsiExtOscChargeCurrent_1uA</i>   | External oscillator charge current is 1uA.   |
| <i>kTsiExtOscChargeCurrent_2uA</i>   | External oscillator charge current is 2uA.   |
| <i>kTsiExtOscChargeCurrent_4uA</i>   | External oscillator charge current is 4uA.   |
| <i>kTsiExtOscChargeCurrent_8uA</i>   | External oscillator charge current is 8uA.   |
| <i>kTsiExtOscChargeCurrent_16uA</i>  | External oscillator charge current is 16uA.  |
| <i>kTsiExtOscChargeCurrent_32uA</i>  | External oscillator charge current is 32uA.  |
| <i>kTsiExtOscChargeCurrent_64uA</i>  | External oscillator charge current is 64uA.  |

### 30.2.3.16 enum tsi\_channel\_number\_t

These bits specify current channel to be measured.

Enumerator

|                            |                   |
|----------------------------|-------------------|
| <i>kTsiChannelNumber_0</i> | Channel Number 0. |
| <i>kTsiChannelNumber_1</i> | Channel Number 1. |
| <i>kTsiChannelNumber_2</i> | Channel Number 2. |
| <i>kTsiChannelNumber_3</i> | Channel Number 3. |
| <i>kTsiChannelNumber_4</i> | Channel Number 4. |
| <i>kTsiChannelNumber_5</i> | Channel Number 5. |
| <i>kTsiChannelNumber_6</i> | Channel Number 6. |
| <i>kTsiChannelNumber_7</i> | Channel Number 7. |
| <i>kTsiChannelNumber_8</i> | Channel Number 8. |

- kTsiChannelNumber\_9* Channel Number 9.
- kTsiChannelNumber\_10* Channel Number 10.
- kTsiChannelNumber\_11* Channel Number 11.
- kTsiChannelNumber\_12* Channel Number 12.
- kTsiChannelNumber\_13* Channel Number 13.
- kTsiChannelNumber\_14* Channel Number 14.
- kTsiChannelNumber\_15* Channel Number 15.

### 30.2.4 Function Documentation

#### 30.2.4.1 void TSI\_HAL\_Init( uint32\_t *baseAddr* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

none

Initialize the peripheral to default state.

#### 30.2.4.2 void TSI\_HAL\_SetConfiguration( uint32\_t *baseAddr*, tsi\_config\_t \* *config* )

Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>baseAddr</i> | TSI module base address.                       |
| <i>config</i>   | Pointer to TSI module configuration structure. |

Returns

none

Initialize and sets prescalers, number of scans, clocks, delta voltage capacitance trimmer, reference and electrode charge current and threshold.

#### 30.2.4.3 uint32\_t TSI\_HAL\_Recalibrate( uint32\_t *baseAddr*, tsi\_config\_t \* *config*, const uint32\_t *electrodes*, const tsi\_parameter\_limits\_t \* *parLimits* )

## TSI HAL driver

Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>baseAddr</i>   | TSI module base address.                          |
| <i>config</i>     | Pointer to TSI module configuration structure.    |
| <i>electrodes</i> | The map of the electrodes.                        |
| <i>parLimits</i>  | Pointer to TSI module parameter limits structure. |

Returns

Lowest signal

This function if TSI basic module is enable, than disable him and if module has enabled interrupt, disable him. Then Set prescaler, electrode and reference current, number of scan and voltage rails. Enable module and interrupt if is not. Better if you see implimetation of this function for better understanding [TSI\\_HAL\\_Recalibrate](#).

### 30.2.4.4 void TSI\_HAL\_EnableLowPower ( uint32\_t *baseAddr* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

none

### 30.2.4.5 void TSI\_HAL\_DisableLowPower ( uint32\_t *baseAddr* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.6 static void TSI\_HAL\_EnableOutOfRangeInterrupt ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### **30.2.4.7 static void TSI\_HAL\_EnableEndOfScanInterrupt ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### **30.2.4.8 static void TSI\_HAL\_EnableModule ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### **30.2.4.9 static void TSI\_HAL\_DisableModule ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

## TSI HAL driver

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.10 static uint32\_t TSI\_HAL\_IsModuleEnabled ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

State of enable module flag.

### 30.2.4.11 static void TSI\_HAL\_EnableStop ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.12 static void TSI\_HAL\_DisableStop ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.13 static void TSI\_HAL\_EnableSoftwareTriggerScan ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

#### 30.2.4.14 static uint32\_t TSI\_HAL\_GetScanTriggerMode ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Scan trigger mode.

#### 30.2.4.15 static uint32\_t TSI\_HAL\_IsScanInProgress ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

True - if scan is in progress. False - otherwise

#### 30.2.4.16 static uint32\_t TSI\_HAL\_GetOutOfRangeFlag ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

## TSI HAL driver

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

State of out of range flag.

**30.2.4.17 static void TSI\_HAL\_ClearOutOfRangeFlag ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.18 static uint32\_t TSI\_HAL\_GetEndOfScanFlag ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Current state of end of scan flag.

**30.2.4.19 static void TSI\_HAL\_ClearEndOfScanFlag ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.20 static void TSI\_HAL\_SetPrescaler ( uint32\_t *baseAddr*, tsi\_electrode\_osc\_prescaler\_t *prescaler* ) [inline], [static]**

Parameters

|                  |                          |
|------------------|--------------------------|
| <i>baseAddr</i>  | TSI module base address. |
| <i>prescaler</i> | Prescaler value.         |

Returns

None.

### 30.2.4.21 static tsi\_electrode\_osc\_prescaler\_t TSI\_HAL\_GetPrescaler ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Prescaler value.

### 30.2.4.22 static void TSI\_HAL\_SetNumberOfScans ( *uint32\_t baseAddr*, *tsi\_n\_consecutive\_scans\_t number* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>number</i>   | Number of scans.         |

Returns

None.

### 30.2.4.23 static tsi\_n\_consecutive\_scans\_t TSI\_HAL\_GetNumberOfScans ( *uint32\_t baseAddr* ) [inline], [static]

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Number of scans.

**30.2.4.24 static void TSI\_HAL\_EnablePeriodicalScan ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.25 static void TSI\_HAL\_EnableErrorInterrupt ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.26 static void TSI\_HAL\_DisableErrorInterrupt ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.27 static void TSI\_HAL\_EnableInterrupt ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.28 static void TSI\_HAL\_DisableInterrupt ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.29 static uint32\_t TSI\_HAL\_IsInterruptEnabled ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

State of enable interrupt flag.

### 30.2.4.30 static void TSI\_HAL\_StartSoftwareTrigger ( uint32\_t *baseAddr* ) [inline], [static]

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.31 static uint32\_t TSI\_HAL\_IsOverrun ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

State of over run flag.

**30.2.4.32 static void TSI\_HAL\_ClearOverrunFlag ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.33 static uint32\_t TSI\_HAL\_GetExternalElectrodeErrorFlag ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Stae of external electrode error flag

30.2.4.34 **static void TSI\_HAL\_ClearExternalElectrodeErrorFlag ( uint32\_t baseAddr )**  
[**inline**], [**static**]

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.35 static void TSI\_HAL\_SetLowPowerScanInterval ( uint32\_t *baseAddr*, tsi\_low\_power\_interval\_t *interval* ) [inline], [static]**

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>baseAddr</i> | TSI module base address.     |
| <i>interval</i> | Interval for low power scan. |

Returns

None.

**30.2.4.36 static tsi\_low\_power\_interval\_t TSI\_HAL\_GetLowPowerScanInterval ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Interval for low power scan.

**30.2.4.37 static void TSI\_HAL\_SetLowPowerClock ( uint32\_t *baseAddr*, uint32\_t *clock* ) [inline], [static]**

Parameters

|                 |                            |
|-----------------|----------------------------|
| <i>baseAddr</i> | TSI module base address.   |
| <i>clock</i>    | Low power clock selection. |

### 30.2.4.38 static uint32\_t TSI\_HAL\_GetLowPowerClock ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Low power clock selection.

### 30.2.4.39 static void TSI\_HAL\_SetReferenceChargeCurrent ( uint32\_t *baseAddr*, tsi\_reference\_osc\_charge\_current\_t *current* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>current</i>  | The charge current.      |

Returns

None.

### 30.2.4.40 static tsi\_reference\_osc\_charge\_current\_t TSI\_HAL\_GetReferenceChargeCurrent ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

The charge current.

### 30.2.4.41 static void TSI\_HAL\_SetElectrodeChargeCurrent ( uint32\_t *baseAddr*, tsi\_external\_osc\_charge\_current\_t *current* ) [inline], [static]

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>current</i>  | Electrode current.       |

Returns

None.

**30.2.4.42 static tsi\_external\_osc\_charge\_current\_t TSI\_HAL\_GetElectrodeChargeCurrent ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Charge current.

**30.2.4.43 static void TSI\_HAL\_SetScanModulo ( uint32\_t *baseAddr*, uint32\_t *modulo* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>modulo</i>   | Scan modulo value.       |

Returns

None.

**30.2.4.44 static uint32\_t TSI\_HAL\_GetScanModulo ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Scan modulo value.

#### 30.2.4.45 static void TSI\_HAL\_SetActiveModeSource ( *uint32\_t baseAddr, uint32\_t source* ) [inline], [static]

Parameters

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>baseAddr</i> | TSI module base address.                                |
| <i>source</i>   | Active mode clock source (LPOSCLK, MCGIRCLK, OSCERCLK). |

Returns

None.

#### 30.2.4.46 static uint32\_t TSI\_HAL\_GetActiveModeSource ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Source value.

#### 30.2.4.47 static void TSI\_HAL\_SetActiveModePrescaler ( *uint32\_t baseAddr, tsi\_active\_mode\_prescaler\_t prescaler* ) [inline], [static]

## TSI HAL driver

Parameters

|                  |                          |
|------------------|--------------------------|
| <i>baseAddr</i>  | TSI module base address. |
| <i>prescaler</i> | Prescaler's value.       |

Returns

None.

**30.2.4.48 static uint32\_t TSI\_HAL\_GetActiveModePrescaler ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Prescaler's value.

**30.2.4.49 static void TSI\_HAL\_SetLowPowerChannel ( uint32\_t *baseAddr*, uint32\_t *channel* ) [inline], [static]**

Only one channel can wake up MCU.

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>channel</i>  | Channel number.          |

Returns

None.

**30.2.4.50 static uint32\_t TSI\_HAL\_GetLowPowerChannel ( uint32\_t *baseAddr* ) [inline], [static]**

Only one channel can wake up MCU.

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Channel number.

### 30.2.4.51 static void TSI\_HAL\_EnableChannel ( *uint32\_t baseAddr, uint32\_t channel* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>channel</i>  | Channel to be enabled.   |

Returns

None.

### 30.2.4.52 static void TSI\_HAL\_EnableChannels ( *uint32\_t baseAddr, uint32\_t channelsMask* ) [inline], [static]

The function enables channels by mask. It can set all at once.

Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>baseAddr</i>     | TSI module base address.     |
| <i>channelsMask</i> | Channels mask to be enabled. |

Returns

None.

### 30.2.4.53 static void TSI\_HAL\_DisableChannel ( *uint32\_t baseAddr, uint32\_t channel* ) [inline], [static]

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>channel</i>  | Channel to be disabled.  |

Returns

None.

### 30.2.4.54 static void TSI\_HAL\_DisableChannels ( *uint32\_t baseAddr, uint32\_t channelsMask* ) [inline], [static]

The function disables channels by mask. It can set all at once.

Parameters

|                     |                               |
|---------------------|-------------------------------|
| <i>baseAddr</i>     | TSI module base address.      |
| <i>channelsMask</i> | Channels mask to be disabled. |

Returns

None.

### 30.2.4.55 static uint32\_t TSI\_HAL\_GetEnabledChannel ( *uint32\_t baseAddr, uint32\_t channel* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>channel</i>  | Channel to be checked.   |

Returns

True - if channel is enabled, false - otherwise.

### 30.2.4.56 static uint32\_t TSI\_HAL\_GetEnabledChannels ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Channels mask that are enabled.

### 30.2.4.57 static uint16\_t TSI\_HAL\_GetWakeUpChannelCounter ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Wake up counter value.

### 30.2.4.58 static uint32\_t TSI\_HAL\_GetCounter ( *uint32\_t baseAddr, uint32\_t channel* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>channel</i>  | Index of TSI channel.    |

Returns

The counter value.

### 30.2.4.59 static void TSI\_HAL\_SetLowThreshold ( *uint32\_t baseAddr, uint32\_t low\_threshold* ) [inline], [static]

## TSI HAL driver

Parameters

|                      |                          |
|----------------------|--------------------------|
| <i>baseAddr</i>      | TSI module base address. |
| <i>low_threshold</i> | Low counter threshold.   |

Returns

None.

### 30.2.4.60 static void TSI\_HAL\_SetHighThreshold ( *uint32\_t baseAddr, uint32\_t high\_threshold* ) [inline], [static]

Parameters

|                       |                          |
|-----------------------|--------------------------|
| <i>baseAddr</i>       | TSI module base address. |
| <i>high_threshold</i> | High counter threshold.  |

Returns

None.

### 30.2.4.61 static uint32\_t TSI\_HAL\_GetEnableStop ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Number of scans.

### 30.2.4.62 static void TSI\_HAL\_SetEnableStop ( *uint32\_t baseAddr* ) [inline], [static]

This enables TSI module function in low power modes.

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### **30.2.4.63 static void TSI\_HAL\_SetDisableStop ( uint32\_t *baseAddr* ) [inline], [static]**

The TSI is disabled in low power modes.

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### **30.2.4.64 static void TSI\_HAL\_EnableHardwareTriggerScan ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### **30.2.4.65 static void TSI\_HAL\_CurrentSourcePairSwapped ( uint32\_t *baseAddr* ) [inline], [static]**

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.66 static void TSI\_HAL\_CurrentSourcePairNotSwapped ( uint32\_t *baseAddr* )  
[inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

**30.2.4.67 static uint32\_t TSI\_HAL\_GetCurrentSourcePairSwapped ( uint32\_t *baseAddr* )  
[inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Current source pair swapped status.

**30.2.4.68 static void TSI\_HAL\_SetMeasuredChannelNumber ( uint32\_t *baseAddr*,  
uint32\_t *channel* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>channel</i>  | Channel number 0 ... 15. |

Returns

None.

30.2.4.69 **static uint32\_t TSI\_HAL\_GetMeasuredChannelNumber ( uint32\_t *baseAddr* )**  
[**inline**], [**static**]

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

uint32\_t Channel number 0 ... 15.

### 30.2.4.70 static void TSI\_HAL\_DmaTransferEnable ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.71 static void TSI\_HAL\_DmaTransferDisable ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

None.

### 30.2.4.72 static uint32\_t TSI\_HAL\_IsDmaTransferEnable ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

State of enable module flag.

30.2.4.73 **static uint32\_t TSI\_HAL\_GetCounter( uint32\_t *baseAddr* ) [inline],  
[static]**

## TSI HAL driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

Accumulated scan counter value ticked by the reference clock.

### 30.2.4.74 static void TSI\_HAL\_SetMode ( uint32\_t *baseAddr*, tsi\_analog\_mode\_select\_t *mode* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>mode</i>     | Mode value.              |

Returns

None.

### 30.2.4.75 static tsi\_analog\_mode\_select\_t TSI\_HAL\_GetMode ( uint32\_t *baseAddr* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

tsi\_analog\_mode\_select\_t Mode value.

### 30.2.4.76 static void TSI\_HAL\_SetOscillatorVoltageRails ( uint32\_t *baseAddr*, tsi\_oscillator\_voltage\_rails\_t *dvolt* ) [inline], [static]

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
| <i>dvolt</i>    | The voltage rails.       |

Returns

None.

### **30.2.4.77 static tsi\_oscillator\_voltage\_rails\_t TSI\_HAL\_GetOscillatorVoltageRails ( uint32\_t *baseAddr* ) [inline], [static]**

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>baseAddr</i> | TSI module base address. |
|-----------------|--------------------------|

Returns

*dvolt* The voltage rails..

## TSI Peripheral Driver

### 30.3 TSI Peripheral Driver

The chapter describes the programming interface of the TSI Peripheral driver.

#### 30.3.1 Overview

#### Files

- file [fsl\\_tsi\\_driver.h](#)

#### Data Structures

- struct [tsi\\_user\\_config\\_t](#)  
*User configuration structure for TSI driver. [More...](#)*
- struct [tsi\\_operation\\_mode\\_t](#)  
*Driver operation mode data hold structure. [More...](#)*
- struct [tsi\\_state\\_t](#)  
*Driver data storage place. [More...](#)*

#### Typedefs

- typedef void(\* [tsi\\_callback\\_t](#)) (uint32\_t instance, void \*userData)  
*Call back routine of TSI driver.*

#### Enumerations

- enum [tsi\\_modes\\_t](#) {  
    [tsi\\_OpModeNormal](#) = 0,  
    [tsi\\_OpModeProximity](#),  
    [tsi\\_OpModeLowPower](#),  
    [tsi\\_OpModeNoise](#),  
    [tsi\\_OpModeCnt](#),  
    [tsi\\_OpModeNoChange](#) }  
*Driver operation mode definition.*

#### Variables

- const uint32\_t [g\\_tsiBaseAddr](#) []  
*Table of base addresses for tsi instances.*
- const IRQn\_Type [g\\_tsiIrqId](#) [HW\_TSI\_INSTANCE\_COUNT]  
*Table to save tsi IRQ enum numbers defined in CMSIS header file.*
- [tsi\\_state\\_t](#) \* [g\\_tsiStatePtr](#) [HW\_TSI\_INSTANCE\_COUNT]  
*Table to save pointers to context data.*

## Initialization

- `tsi_status_t TSI_DRV_Init (const uint32_t instance, tsi_state_t *tsiState, const tsi_user_config_t *tsiUserConfig)`

*This function initializes a TSI instance for operation.*
- `tsi_status_t TSI_DRV_DeInit (const uint32_t instance)`

*This function shuts down the TSI by disabling interrupts and disable the peripheral itself.*
- `tsi_status_t TSI_DRV_EnableElectrode (const uint32_t instance, const uint32_t channel, const bool enable)`

*This function enables/disables one electrode of TSI module.*
- `uint32_t TSI_DRV_GetEnabledElectrodes (const uint32_t instance)`

*This function return mask of enabled electrodes of TSI module.*
- `tsi_status_t TSI_DRV_Measure (const uint32_t instance)`

*This function starts the measure cycle of enabled electrodes.*
- `tsi_status_t TSI_DRV_MeasureBlocking (const uint32_t instance)`

*This function starts the measure cycle of enabled electrodes in blocking mode.*
- `tsi_status_t TSI_DRV_AbortMeasure (const uint32_t instance)`

*This function aborts the measure cycle in non standard situation use of driver.*
- `tsi_status_t TSI_DRV_GetCounter (const uint32_t instance, const uint32_t channel, uint16_t *counter)`

*This function return the last value of measurement values.*
- `tsi_status_t TSI_DRV_GetStatus (const uint32_t instance)`

*This function return the current status of the driver.*
- `tsi_status_t TSI_DRV_EnableLowPower (const uint32_t instance)`

*This function enters to low power mode of TSI driver.*
- `tsi_status_t TSI_DRV_DisableLowPower (const uint32_t instance, const tsi_modes_t mode)`

*This function returns back the TSI driver from the low power to standard operation.*
- `tsi_status_t TSI_DRV_Recalibrate (const uint32_t instance, uint32_t *lowestSignal)`

*This function do automatic recalibration of all important setting of TSI.*
- `tsi_status_t TSI_DRV_SetCallBackFunc (const uint32_t instance, const tsi_callback_t pFuncCallBack, void *userData)`

*This function sets the callback function that is called when the measure cycle ends.*
- `tsi_status_t TSI_DRV_ChangeMode (const uint32_t instance, const tsi_modes_t mode)`

*This function change the current working operation mode.*
- `tsi_modes_t TSI_DRV_GetMode (const uint32_t instance)`

*This function returns the current working operation mode.*
- `tsi_status_t TSI_DRV_LoadConfiguration (const uint32_t instance, const tsi_modes_t mode, const tsi_operation_mode_t *operationMode)`

*This function load to TSI driver configuration for specific mode.*
- `tsi_status_t TSI_DRV_SaveConfiguration (const uint32_t instance, const tsi_modes_t mode, tsi_operation_mode_t *operationMode)`

*This function save the TSI driver configuration for specific mode.*

## TSI Peripheral Driver

### 30.3.1.1 TSI Peripheral Driver

#### Overview

The TSI peripheral driver provide functionality to measure capacitance on electrodes (touch sensing) in various modes and provides APIs for input/output operations.

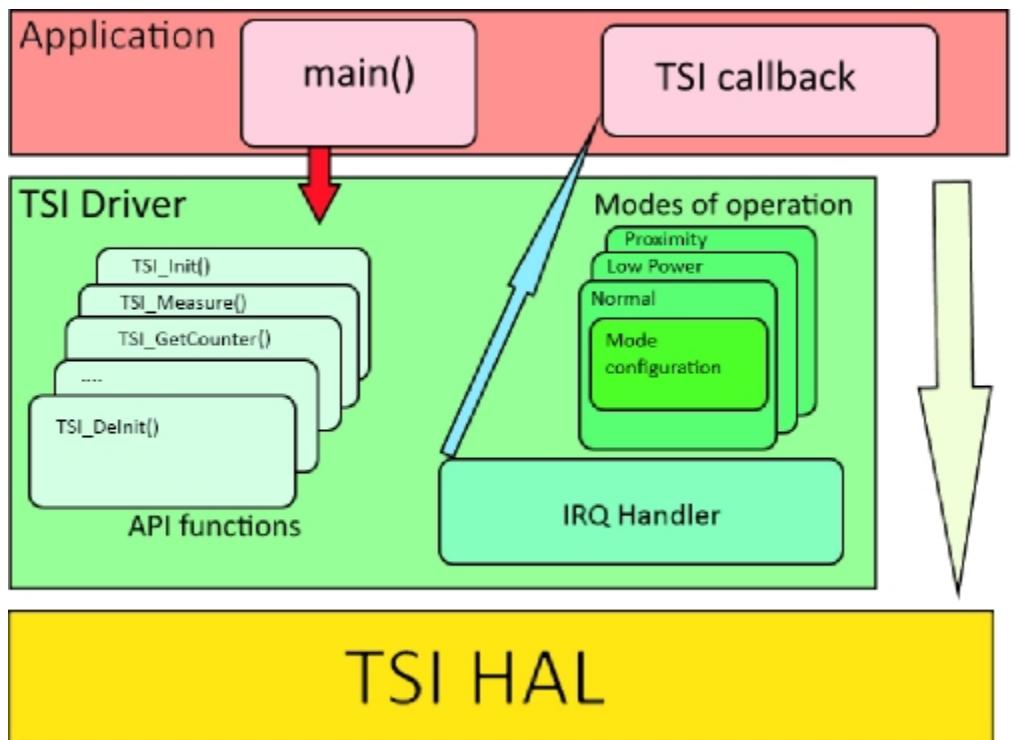


Figure 30.3.1: TSI Driver Block Diagram

#### TSI Initialization

The initialization of the TSI driver is simple, the application must define the startup configuration and handle to initialization routine.

Include "fsl\_tsi\_driver.h" header file in source files where you want to use TSI driver to operate touch sensing functionality.

Example of the TSI driver initialization (with example of configuration) and simple measurement:

```
#include "fsl_tsi_driver.h"

// Declaration of TSI callback function
static void tsi_irq_callback(uint32_t instance, void* usrData);

// Set up the HW configuration for normal mode of TSI
static const tsi_config_t tsi_HwConfig =
```

```

{
 .ps = kTsiElecOscPrescaler_2div,
 .extchrg = kTsiExtOscChargeCurrent_10uA,
 .refchrg = kTsiRefOscChargeCurrent_10uA,
 .nscn = kTsiConsecutiveScansNumber_8time,
 .lpciks = kTsiLowPowerInterval_100ms,
 .amciks = kTsiActiveClkSource_BusClock,
 .ampsc = kTsiActiveModePrescaler_8div,
 .lpscnity = kTsiLowPowerInterval_100ms,
 .thresh = 100,
 .thresl = 200,
};

// Set up the configuration of initialization structure
static const tsi_user_config_t myTsiConfig =
{
 .config = (tsi_config_t*)&tsi_HwConfig,
 .pCallBackFunc = tsi_irq_callback, // My
 .usrData = (void*)0x12345678,
};

// Flag the the measure of TSI ends
static bool tsi_measureEnds;

// The measured value
static uint16_t tsiData;

// The definition of channel number
#define BOARD_TSI_ELECTRODE_1 1

void main(void)
{
 tsi_status_t result;

 result = TSI_DRV_Init(0, &myTsiState, &myTsiConfig);

 if(result != kStatus_TSI_Success)
 {
 printf("\nThe initialization of TSI driver failed. Result is: %d.", (uint32_t)result);
 }

 // Enable electrodes for normal mode
 result = TSI_DRV_EnableElectrode(0, BOARD_TSI_ELECTRODE_1, true);

 if(result != kStatus_TSI_Success)
 {
 printf("\nThe initialization of TSI channel failed. Result is: %d.", (uint32_t)result);
 }

 tsi_measureEnds = false;

 // Kick of measuring process
 result = TSI_DRV_Measure(0);

 if(result != kStatus_TSI_Success)
 {
 printf("\nThe call measure of TSI failed. Result is: %d.", (uint32_t)result);
 }

 while(1)
 {
 if(tsi_measureEnds == true)
 {
 // new TSI data are ready to read
 result = TSI_DRV_GetCounter(0, BOARD_TSI_ELECTRODE_1, &tsiData);

 if(result != kStatus_TSI_Success)
 {

```

## TSI Peripheral Driver

```
 printf("\nThe read of TSI failed. Result is: %d.", (uint32_t)result);
}

// And measure again
result = TSI_DRV_Measure(0);

if(result != kStatus_TSI_Success)
{
 printf("\nThe call measure of TSI failed. Result is: %d.", (uint32_t)result);
}else
{
 tsi_measureEnds = false;
}
}

/*
 * Call back function of TSI driver */
static void tsi_irq_callback(uint32_t instance, void* usrData)
{
 // The measure cycle ends
 tsi_measureEnds = true;
}
```

### 30.3.2 Data Structure Documentation

#### 30.3.2.1 struct tsi\_user\_config\_t

Use an instance of this structure with [TSI\\_DRV\\_Init\(\)](#). This allows you to configure the most common settings of the TSI peripheral with a single function call. Settings include:

##### Data Fields

- `tsi_config_t * config`
- `tsi_callback_t pCallBackFunc`
- `void * userData`

##### 30.3.2.1.0.62 Field Documentation

###### 30.3.2.1.0.62.1 `tsi_config_t* tsi_user_config_t::config`

A pointer to hardware configuration. Can't be NULL.

###### 30.3.2.1.0.62.2 `tsi_callback_t tsi_user_config_t::pCallBackFunc`

A pointer to call back function of end of mesurement.

###### 30.3.2.1.0.62.3 `void* tsi_user_config_t::userData`

A user data of call back function.

### 30.3.2.2 struct tsi\_operation\_mode\_t

This is operation mode data hold structure. The structure is keep all needed data to be driver able to switch the operation modes and properly set up HW peripheral.

#### Data Fields

- uint16\_t enabledElectrodes
- tsi\_config\_t config

#### 30.3.2.2.0.63 Field Documentation

##### 30.3.2.2.0.63.1 uint16\_t tsi\_operation\_mode\_t::enabledElectrodes

The back up of enabled electrodes for operation mode

##### 30.3.2.2.0.63.2 tsi\_config\_t tsi\_operation\_mode\_t::config

A hardware configuration.

### 30.3.2.3 struct tsi\_state\_t

It must be created by application code and the pointer is handled by [TSI\\_DRV\\_Init](#) function to driver. The driver keeps all contex data for itself run. Settings include:

#### Data Fields

- tsi\_status\_t status
- tsi\_callback\_t pCallBackFunc
- void \*usrData
- semaphore\_t irqSync
- mutex\_t lock
- mutex\_t lockChangeMode
- bool isBlockingMeasure
- tsi\_modes\_t opMode
- tsi\_operation\_mode\_t opModesData [tsi\_OpModeCnt]
- uint16\_t counters [FSL\_FEATURE\_TSI\_CHANNEL\_COUNT]

#### 30.3.2.3.0.64 Field Documentation

##### 30.3.2.3.0.64.1 tsi\_status\_t tsi\_state\_t::status

Current status of the driver.

##### 30.3.2.3.0.64.2 tsi\_callback\_t tsi\_state\_t::pCallBackFunc

A pointer to call back function of end of measurement.

## TSI Peripheral Driver

### 30.3.2.3.0.64.3 `void* tsi_state_t::userData`

A user data pointer handled by call back function.

### 30.3.2.3.0.64.4 `semaphore_t tsi_state_t::irqSync`

Used to wait for ISR to complete its measure business.

### 30.3.2.3.0.64.5 `mutex_t tsi_state_t::lock`

Used by whole driver to secure the context data integrity.

### 30.3.2.3.0.64.6 `mutex_t tsi_state_t::lockChangeMode`

Used by chenge mode function to secure the context data integrity.

### 30.3.2.3.0.64.7 `bool tsi_state_t::isBlockingMeasure`

Used to internal indication of type of measurement.

### 30.3.2.3.0.64.8 `tsi_modes_t tsi_state_t::opMode`

Storage of current operation mode.

### 30.3.2.3.0.64.9 `tsi_operation_mode_t tsi_state_t::opModesData[tsi_OpModeCnt]`

Data storage of individual operational modes.

### 30.3.2.3.0.64.10 `uint16_t tsi_state_t::counters[FSL_FEATURE_TSI_CHANNEL_COUNT]`

The mirrow of last state of counter registers

## 30.3.3 Typedef Documentation

### 30.3.3.1 `typedef void(* tsi_callback_t)(uint32_t instance, void *userData)`

The function is called on end of the measure of the TSI driver. The function can be called from interrupt, so the code inside the callback should be short and fast.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>instance</i> | - instance of the TSI peripheral |
|-----------------|----------------------------------|

|                 |                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------|
| <i>userData</i> | - user data (type is void*), the user data are specified by function <a href="#">TSI_DRV_SetCallBackFunc</a> |
|-----------------|--------------------------------------------------------------------------------------------------------------|

Returns

- none

### 30.3.4 Enumeration Type Documentation

#### 30.3.4.1 enum tsi\_modes\_t

The operation name definition used for TSI driver.

Enumerator

- tsi\_OpModeNormal* The normal mode of TSI.
- tsi\_OpModeProximity* The proximity sensing mode of TSI.
- tsi\_OpModeLowPower* The low power mode of TSI.
- tsi\_OpModeNoise* The noise mode of TSI. This mode is not valid with TSI HW, valid only for TSIL HW.
- tsi\_OpModeCnt* Count of TSI modes - for internal use.
- tsi\_OpModeNoChange* The special value of operation mode that allows call for example [TSI\\_DRV\\_DisableLowPower](#) function without change of operation mode.

### 30.3.5 Function Documentation

#### 30.3.5.1 tsi\_status\_t TSI\_DRV\_Init ( const uint32\_t *instance*, tsi\_state\_t \* *tsiState*, const tsi\_user\_config\_t \* *tsiUserConfig* )

This function will initialize the run-time state structure and prepare the whole peripheral for measure the capacitances on electrodes.

```
static tsi_state_t myTsiDriverStateStructure;

tsi_user_config_t myTsiDriveruserConfig =
{
 .config =
 {
 ...
 },
 .pCallBackFunc = APP_myTsiCallBackFunc,
 .usrData = myData,
};

if(TSI_DRV_Init(0, &myTsiDriverStateStructure, &myTsiDriveruserConfig) != kStatus_TSI_Success)
{
 // Error, the TSI is not initialized
}
```

## TSI Peripheral Driver

Parameters

|                      |                                                                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>      | The TSI module instance.                                                                                                                                                                                                                                                                           |
| <i>tsiState</i>      | A pointer to the TSI driver state structure memory. The user is only responsible to pass in the memory for this run-time state structure where the TSI driver will take care of filling out the members. This run-time state structure keeps track of the current TSI peripheral and driver state. |
| <i>tsiUserConfig</i> | The user configuration structure of type <a href="#">tsi_user_config_t</a> . The user is responsible to fill out the members of this structure and to pass the pointer of this struct into this function.                                                                                          |

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.2 tsi\_status\_t TSI\_DRV\_DeInit ( const uint32\_t *instance* )

This function disables the TSI interrupts and disables the peripheral.

```
if(TSI_DRV_DeInit(0) != kStatus_TSI_Success)
{
 // Error, the TSI is not deinitialized
}
```

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.3 tsi\_status\_t TSI\_DRV\_EnableElectrode ( const uint32\_t *instance*, const uint32\_t *channel*, const bool *enable* )

Function must be called for each used electrodes after initialization of TSI driver.

```
// On the TSI instance 0, enable electrode with index 5
if(TSI_DRV_EnableElectrode(0, 5, TRUE) != kStatus_TSI_Success)
{
 // Error, the TSI 5'th electrode is not enabled
}
```

## Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>instance</i> | The TSI module instance.                      |
| <i>channel</i>  | Index of TSI channel.                         |
| <i>enable</i>   | TRUE - for channel enable, FALSE for disable. |

## Returns

An error code or kStatus\_TSI\_Success.

**30.3.5.4 uint32\_t TSI\_DRV\_GetEnabledElectrodes ( const uint32\_t *instance* )**

Function returns mask of the enabled electrodes of current mode.

```
uint32_t enabledElectrodeMask;
enabledElectrodeMask = TSI_DRV_GetEnabledElectrodes(0);
```

## Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

## Returns

Mask of enebed electrodes for current mode.

**30.3.5.5 tsi\_status\_t TSI\_DRV\_Measure ( const uint32\_t *instance* )**

Function is non blocking, so for the result must be wait after driver finish the measure cycle. The end of measure cycle can be checked by pooling [TSI\\_DRV\\_GetStatus](#) function or wait for registered callback function by [TSI\\_DRV\\_SetCallBackFunc](#) or [TSI\\_DRV\\_Init](#).

```
// Example of pooling style of use of TSI_DRV_Measure() function
if(TSI_DRV_Measure(0) != kStatus_TSI_Success)
{
 // Error, the TSI 5'th electrode is not enabled
}

while(TSI_DRV_GetStatus(0) != kStatus_TSI_Initialized)
{
 // Do something useful - don't waste the CPU cycle time
}
```

## TSI Peripheral Driver

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.6 **tsi\_status\_t TSI\_DRV\_MeasureBlocking ( const uint32\_t *instance* )**

Function is blocking, so the after finish this function call, the result of measured electrodes is available immediately and can be get by [TSI\\_DRV\\_GetCounter](#) function.

```
// Example of pooling style of use of TSI_DRV_Measure() function
if(TSI_DRV_MeasureBlocking(0) != kStatus_TSI_Success)
{
 // Error, the TSI 5'th electrode is not enabled
} else
{
 // Get the result by TSI_DRV_GetCounter function
}
```

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.7 **tsi\_status\_t TSI\_DRV\_AbortMeasure ( const uint32\_t *instance* )**

Function aborts currently running measure cycle, and it's designed for unexpected situations. Is not targeted for standard use.

```
// Start measure by @ref TSI_DRV_Measure() function
if(TSI_DRV_Measure(0) != kStatus_TSI_Success)
{
 // Error, the TSI 5'th electrode is not enabled
}

if(isNeededAbort) // I need abort measure from any application reason
{
 TSI_DRV_AbortMeasure(0);
}
```

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.8 **tsi\_status\_t TSI\_DRV\_GetCounter ( const uint32\_t *instance*, const uint32\_t *channel*, uint16\_t \* *counter* )**

Function return the last measure values of previous measure cycle of driver the data are buffered inside of driver.

```
// Get the counter value from TSI instance 0 and 5th channel
uint32_t result;
if(TSI_DRV_GetCounter(0, 5, &result) != kStatus_TSI_Success)
{
 // Error, the TSI 5'th electrode is not readed
}
```

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>instance</i> | The TSI module instance.                                                |
| <i>channel</i>  | The TSI electrode index.                                                |
| <i>counter</i>  | The pointer to 16 bit value where will be stored channel counter value. |

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.9 **tsi\_status\_t TSI\_DRV\_GetStatus ( const uint32\_t *instance* )**

Function return the current working status of the driver.

```
// Get the current status of TSI driver
tsi_status_t status;
status = TSI_DRV_GetStatus(0);
```

## TSI Peripheral Driver

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

### Returns

An current status of the driver.

### 30.3.5.10 **tsi\_status\_t TSI\_DRV\_EnableLowPower ( const uint32\_t *instance* )**

Function switch the driver to low power mode and immediately enable the low power functionality of TSI peripheral. Before calling this function the low power mode must be configured - Enabled right electrode and recalibrate the low power mode to get best performance for this mode.

```
// Switch the driver to the low power mode
uint16_t signal;

// For fisrt time is needed configure the low power mode configuration
(void)TSI_DRV_ChangeMode(0, tsi_OpModeLowPower); // I don't check the
 result because I believe in.
// Enable the right one electrode for low power ake up operation
(void)TSI_DRV_EnableElectrode(0, 5, true);
// Recalibrate the mode to get the best performance for this one electrode
(void)TSI_DRV_Recalibrate(0, &signal);

if(TSI_DRV_EnableLowPower(0) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't go to low power mode
}
```

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

### Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.11 **tsi\_status\_t TSI\_DRV\_DisableLowPower ( const uint32\_t *instance*, const tsi\_modes\_t *mode* )**

Function switch the driver back form low power mode and it can immediately change the operation mode to any other or keep the driver in low power configuration, to be able go back to low power state.

```
// Switch the driver from the low power mode

if(TSI_DRV_DisableLowPower(0, tsi_OpModeNormal) !=
 kStatus_TSI_Success)
{
 // Error, the TSI driver can't go from low power mode
}
```

## Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | The TSI module instance.       |
| <i>mode</i>     | The new operation mode request |

## Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.12 tsi\_status\_t TSI\_DRV\_Recalibrate ( const uint32\_t *instance*, uint32\_t \* *lowestSignal* )

Function force the the driver to start the recalibration procedure for current operation mode to get best possible TSI hardware settings. The computed setting will be store into the operation mode data and can be Load & Save by [TSI\\_DRV\\_LoadConfiguration](#) or [TSI\\_DRV\\_SaveConfiguration](#) functions.

## Warning

The function could take bigger amount of time (it cannot be specified exact value) and is blocking.

```
// Recalibrate current mode
uint16_t signal;

// Recalibrate the mode to get the best performance for this one electrode

if(TSI_DRV_Recalibrate(0, &signal) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't recalibrate this mode
}
```

## Parameters

|                     |                                                                                 |
|---------------------|---------------------------------------------------------------------------------|
| <i>instance</i>     | The TSI module instance.                                                        |
| <i>lowestSignal</i> | The pointer to variable where will be store the lowest signal of all electrodes |

## Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.13 tsi\_status\_t TSI\_DRV\_SetCallBackFunc ( const uint32\_t *instance*, const tsi\_callback\_t *pFuncCallBack*, void \* *userData* )

Function set up or clear (parameter pFuncCallBack = NULL), the callback function pointer which will be called after each measure cycle ends. The user can also set own user data, that will be handled by parameter to call back function (can be used to call one function by more sources).

## TSI Peripheral Driver

```
// Clear previous call back function

if(TSI_DRV_SetCallBackFunc(0, NULL, NULL) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't set up the call back function at the momment
}

// Set new call back function

if(TSI_DRV_SetCallBackFunc(0, myFunction, (void*)0x12345678) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't set up the call back function at the momment
}
```

### Parameters

|                      |                                               |
|----------------------|-----------------------------------------------|
| <i>instance</i>      | The TSI module instance.                      |
| <i>pFuncCallBack</i> | The pointer to application call back function |
| <i>userData</i>      | The user data pointer                         |

### Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.14 tsi\_status\_t TSI\_DRV\_ChangeMode ( const uint32\_t *instance*, const tsi\_modes\_t *mode* )

Function change the working operation mode of driver.

```
// Change operation mode to low power

if(TSI_DRV_ChangeMode(0, tsi_OpModeLowPower) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't change the operation mode into low power
}
```

### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>instance</i> | The TSI module instance.         |
| <i>mode</i>     | The requested new operation mode |

### Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.15 **`tsi_modes_t TSI_DRV_GetMode ( const uint32_t instance )`**

Function returns the current working operation mode of driver.

```
// Gets current operation mode of TSI driver
tsi_modes_t mode;

mode = TSI_DRV_GetMode(0);
```

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An current operation mode of TSI driver.

### 30.3.5.16 **`tsi_status_t TSI_DRV_LoadConfiguration ( const uint32_t instance, const tsi_modes_t mode, const tsi_operation_mode_t * operationMode )`**

Function loads the new configuration into specific mode. This can be used when the calibrated data are store in any NVM to load after startup of MCU, to avoid run recalibration what takes a bigger amount of time.

```
// Load operation mode configuration

extern const tsi_operation_mode_t * myTsiNvmLowPowerConfiguration;

if(TSI_DRV_LoadConfiguration(0, tsi_OpModeLowPower,
 myTsiNvmLowPowerConfiguration) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't load the configuration
}
```

Parameters

|                      |                                                                         |
|----------------------|-------------------------------------------------------------------------|
| <i>instance</i>      | The TSI module instance.                                                |
| <i>mode</i>          | The requested new operation mode                                        |
| <i>operationMode</i> | The pointer to storage place of the configuration that should be loaded |

Returns

An error code or kStatus\_TSI\_Success.

### 30.3.5.17 **tsi\_status\_t TSI\_DRV\_SaveConfiguration ( const uint32\_t *instance*, const tsi\_modes\_t *mode*, tsi\_operation\_mode\_t \* *operationMode* )**

Function saves the configuration of specific mode. This can be used when the calibrated data should be stored in any backup memory to load after startup of MCU, to avoid run recalibration what takes a bigger amount of time.

```
// Save operation mode configuration

extern tsi_operation_mode_t myTsiNvmLowPowerConfiguration;

if(TSI_DRV_SaveConfiguration(0, tsi_OpModeLowPower, &
 myTsiNvmLowPowerConfiguration) != kStatus_TSI_Success)
{
 // Error, the TSI driver can't save the configuration
}
```

#### Parameters

|                      |                                                                       |
|----------------------|-----------------------------------------------------------------------|
| <i>instance</i>      | The TSI module instance.                                              |
| <i>mode</i>          | The requested new operation mode                                      |
| <i>operationMode</i> | The pointer to storage place of the configuration that should be save |

#### Returns

An error code or kStatus\_TSI\_Success.

## 30.3.6 Variable Documentation

### 30.3.6.1 **const uint32\_t g\_tsibaseAddr[]**

### 30.3.6.2 **const IRQn\_Type g\_tsilrqld[HW\_TSI\_INSTANCE\_COUNT]**

### 30.3.6.3 **tsi\_state\_t\* g\_tsistateptr[HW\_TSI\_INSTANCE\_COUNT]**

# **Chapter 31**

## **Watchdog Timer (WDOG)**

### **31.1 Overview**

The Kinetis SDK provides both HAL and Peripheral drivers for the Watchdog Timer (WDOG) block of Kinetis devices.

### **Modules**

- WDOG HAL driver
- WDOG Peripheral driver

## WDOG HAL driver

### 31.2 WDOG HAL driver

#### 31.2.1 Overview

This section describes the programming interface of the WDOG HAL driver.

## Data Structures

- union `wdog_common_config`  
*Define the common config. [More...](#)*

## Enumerations

- enum `wdog_clock_prescaler_value_t` {  
    kWdogClockPrescalerValueDevide1 = 0x0U,  
    kWdogClockPrescalerValueDevide2 = 0x1U,  
    kWdogClockPrescalerValueDevide3 = 0x2U,  
    kWdogClockPrescalerValueDevide4 = 0x3U,  
    kWdogClockPrescalerValueDevide5 = 0x4U,  
    kWdogClockPrescalerValueDevide6 = 0x5U,  
    kWdogClockPrescalerValueDevide7 = 0x6U,  
    kWdogClockPrescalerValueDevide8 = 0x7U }

*Define the selection of the clock prescaler.*

## Watchdog HAL.

- static void `WDOG_HAL_SetCommonConfig` (uint32\_t baseAddr, `wdog_common_config` commonConfig)  
*Sets the WDOG common config.*
- static void `WDOG_HAL_Enable` (uint32\_t baseAddr)  
*Enables the Watchdog module.*
- static void `WDOG_HAL_Disable` (uint32\_t baseAddr)  
*Disables the Watchdog module.*
- static bool `WDOG_HAL_IsEnabled` (uint32\_t baseAddr)  
*Checks whether the WDOG is enabled.*
- static void `WDOG_HAL_SetIntCmd` (uint32\_t baseAddr, bool enable)  
*Enables and disables the Watchdog interrupt.*
- static bool `WDOG_HAL_GetIntCmd` (uint32\_t baseAddr)  
*Checks whether the WDOG interrupt is enabled.*
- static void `WDOG_HAL_SetClockSourceMode` (uint32\_t baseAddr, uint8\_t clockSource)  
*Sets the Watchdog clock Source.*
- static uint8\_t `WDOG_HAL_GetClockSourceMode` (uint32\_t baseAddr)  
*Gets the Watchdog clock Source.*
- static void `WDOG_HAL_SetWindowModeCmd` (uint32\_t baseAddr, bool enable)  
*Enables and disables the Watchdog window mode.*

- static bool **WDOG\_HAL\_GetWindowModeCmd** (uint32\_t baseAddr)  
*Checks whether the window mode is enabled.*
- static void **WDOG\_HAL\_SetRegisterUpdateCmd** (uint32\_t baseAddr, bool enable)  
*Enables and disables the Watchdog write-once-only register update.*
- static bool **WDOG\_HAL\_GetRegisterUpdateCmd** (uint32\_t baseAddr)  
*Checks whether the register update is enabled.*
- static void **WDOG\_HAL\_SetWorkInDebugModeCmd** (uint32\_t baseAddr, bool enable)  
*Sets whether Watchdog is working while the CPU is in debug mode.*
- static void **WDOG\_HAL\_GetWorkInDebugModeCmd** (uint32\_t baseAddr)  
*Checks whether the WDOG works while in the CPU debug mode.*
- static void **WDOG\_HAL\_SetWorkInStopModeCmd** (uint32\_t baseAddr, bool enable)  
*Sets whether the Watchdog is working while the CPU is in stop mode.*
- static bool **WDOG\_HAL\_GetWorkInStopModeCmd** (uint32\_t baseAddr)  
*Checks whether the WDOG works while in CPU stop mode.*
- static void **WDOG\_HAL\_SetWorkInWaitModeCmd** (uint32\_t baseAddr, bool enable)  
*Sets whether the Watchdog is working while the CPU is in wait mode.*
- static bool **WDOG\_HAL\_GetWorkInWaitModeCmd** (uint32\_t baseAddr)  
*Checks whether the WDOG works while in the CPU wait mode.*
- static bool **WDOG\_HAL\_IsIntPending** (uint32\_t baseAddr)  
*Gets the Watchdog interrupt status.*
- static void **WDOG\_HAL\_ClearIntFlag** (uint32\_t baseAddr)  
*Clears the Watchdog interrupt flag.*
- static void **WDOG\_HAL\_SetTimeoutValue** (uint32\_t baseAddr, uint32\_t timeoutCount)  
*Set the Watchdog timeout value.*
- static uint32\_t **WDOG\_HAL\_GetTimeoutValue** (uint32\_t baseAddr)  
*Gets the Watchdog timeout value.*
- static uint32\_t **WDOG\_HAL\_GetTimerOutputValue** (uint32\_t baseAddr)  
*Gets the Watchdog timer output.*
- static void **WDOG\_HAL\_SetClockPrescalerValueMode** (uint32\_t baseAddr, wdog\_clock\_prescaler\_value\_t clockPrescaler)  
*Sets the Watchdog clock prescaler.*
- static wdog\_clock\_prescaler\_value\_t **WDOG\_HAL\_GetClockPrescalerValueMode** (uint32\_t baseAddr)  
*Gets the Watchdog clock prescaler.*
- static void **WDOG\_HAL\_SetWindowValue** (uint32\_t baseAddr, uint32\_t windowValue)  
*Sets the Watchdog window value.*
- static uint32\_t **WDOG\_HAL\_GetWindowValue** (uint32\_t baseAddr)  
*Gets the Watchdog window value.*
- static void **WDOG\_HAL\_Unlock** (uint32\_t baseAddr)  
*Unlocks the Watchdog register written.*
- static void **WDOG\_HAL\_Refresh** (uint32\_t baseAddr)  
*Refreshes the Watchdog timer.*
- static void **WDOG\_HAL\_ResetSystem** (uint32\_t baseAddr)  
*Resets the chip using the Watchdog.*
- static uint32\_t **WDOG\_HAL\_GetResetCount** (uint32\_t baseAddr)  
*Gets the chip reset count that was reset by Watchdog.*
- static void **WDOG\_HAL\_ClearResetCount** (uint32\_t baseAddr)  
*Clears the chip reset count that was reset by Watchdog.*
- void **WDOG\_HAL\_Init** (uint32\_t baseAddr)  
*Restores the WDOG module to reset value.*

## WDOG HAL driver

### 31.2.2 Data Structure Documentation

#### 31.2.2.1 union wdog\_common\_config

### 31.2.3 Enumeration Type Documentation

#### 31.2.3.1 enum wdog\_clock\_prescaler\_value\_t

Enumerator

|                                        |               |
|----------------------------------------|---------------|
| <i>kWdogClockPrescalerValueDevide1</i> | Divided by 1. |
| <i>kWdogClockPrescalerValueDevide2</i> | Divided by 2. |
| <i>kWdogClockPrescalerValueDevide3</i> | Divided by 3. |
| <i>kWdogClockPrescalerValueDevide4</i> | Divided by 4. |
| <i>kWdogClockPrescalerValueDevide5</i> | Divided by 5. |
| <i>kWdogClockPrescalerValueDevide6</i> | Divided by 6. |
| <i>kWdogClockPrescalerValueDevide7</i> | Divided by 7. |
| <i>kWdogClockPrescalerValueDevide8</i> | Divided by 8. |

### 31.2.4 Function Documentation

#### 31.2.4.1 static void WDOG\_HAL\_SetCommonConfig ( uint32\_t *baseAddr*, wdog\_common\_config *commonConfig* ) [inline], [static]

This function is used to set the WDOG common configure. Make sure WDOG registers are unlocked by the WDOG\_HAL\_Unlock, the WCT window is still open and the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure that the WDOG\_STCTRLH.ALLOW\_UPDATE is 1 which means that the register update is enabled. The common configuration is controlled by the WDOG\_STCTRLH. This is a write-once register and this interface is used to set all field of the WDOG\_STCTRLH registers at the same time. If only one field needs to be set, the API can be used. These API write to the WDOG\_STCTRLH register: [WDOG\\_HAL\\_Enable](#), [WDOG\\_HAL\\_Disable](#), [WDOG\\_HAL\\_SetIntCmd](#), [WDOG\\_HAL\\_SetClockSourceMode](#), [WDOG\\_HAL\\_SetWindowModeCmd](#), [WDOG\\_HAL\\_SetRegisterUpdateCmd](#), [WDOG\\_HAL\\_SetWorkInDebugModeCmd](#), [WDOG\\_HAL\\_SetWorkInStopModeCmd](#), [WDOG\\_HAL\\_SetWorkInWaitModeCmd](#)

Parameters

|                     |                                  |
|---------------------|----------------------------------|
| <i>baseAddr</i>     | The WDOG peripheral base address |
| <i>commonConfig</i> | The common configure of the WDOG |

**31.2.4.2 static void WDOG\_HAL\_Enable ( uint32\_t *baseAddr* ) [inline], [static]**

This function enables the WDOG. Make sure that the WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called.

Parameters

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

**31.2.4.3 static void WDOG\_HAL\_Disable ( uint32\_t *baseAddr* ) [inline], [static]**

This function disables the WDOG. Make sure that the WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called.

Parameters

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

**31.2.4.4 static bool WDOG\_HAL\_IsEnabled ( uint32\_t *baseAddr* ) [inline], [static]**

This function checks whether the WDOG is enabled.

Parameters

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

Returns

false means WDOG is disabled, true means WODG is enabled.

**31.2.4.5 static void WDOG\_HAL\_SetIntCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

This function enables or disables the WDOG interrupt. Make sure that the WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOW\_UPDATE is 1 which means register update is enabled.

## WDOG HAL driver

Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>baseAddr</i> | The WDOG peripheral base address                                                 |
| <i>enable</i>   | false means disable watchdog interrupt and true means enable watchdog interrupt. |

### 31.2.4.6 static bool WDOG\_HAL\_GetIntCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function checks whether the WDOG interrupt is enabled.

Parameters

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

Returns

false means interrupt is disabled, true means interrupt is enabled.

### 31.2.4.7 static void WDOG\_HAL\_SetClockSourceMode ( uint32\_t *baseAddr*, uint8\_t *clockSource* ) [inline], [static]

This function sets the WDOG clock source. There are two clock sources that can be used: the LPO clock and the bus clock. Make sure that the WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                    |                                  |
|--------------------|----------------------------------|
| <i>baseAddr</i>    | The WDOG peripheral base address |
| <i>clockSource</i> | watchdog clock source.           |

### 31.2.4.8 static uint8\_t WDOG\_HAL\_GetClockSourceMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the WDOG clock source. There are two clock sources that can be used: the LPO clock and the bus clock. A Clock Switching Delay time is about 2 clock A cycles plus 2 clock B, where clock A and B are the two input clocks to the clock mux.

Parameters

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

Returns

watchdog clock source.

### 31.2.4.9 static void WDOG\_HAL\_SetWindowModeCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function configures the WDOG window mode. Make sure WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                 |                                                                                   |
|-----------------|-----------------------------------------------------------------------------------|
| <i>baseAddr</i> | The WDOG peripheral base address                                                  |
| <i>enable</i>   | false means disable watchdog window mode. true means enable watchdog window mode. |

### 31.2.4.10 static bool WDOG\_HAL\_GetWindowModeCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function checks whether the WDOG window mode is enabled.

Parameters

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

Returns

false means window mode is disabled, true means window mode is enabled.

### 31.2.4.11 static void WDOG\_HAL\_SetRegisterUpdateCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function configures the WDOG register update feature. If disabled, it means that all WDOG registers is never written again unless Power On Reset. Make sure WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

## WDOG HAL driver

Parameters

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | The WDOG peripheral base address                                                                                          |
| <i>enable</i>   | false means disable watchdog write-once-only register update. true means enable watchdog write-once-only register update. |

### 31.2.4.12 static bool WDOG\_HAL\_GetRegisterUpdateCmd ( *uint32\_t baseAddr* ) [**inline**], [**static**]

This function checks whether the WDOG register update is enabled.

Parameters

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

Returns

false means register update is disabled, true means register update is enabled.

### 31.2.4.13 static void WDOG\_HAL\_SetWorkInDebugModeCmd ( *uint32\_t baseAddr, bool enable* ) [**inline**], [**static**]

This function configures whether the WDOG is enabled in the CPU debug mode. Make sure WDO-G registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                 |                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | The WDOG peripheral base address                                                                      |
| <i>enable</i>   | false means watchdog is disabled in CPU debug mode. true means watchdog is enabled in CPU debug mode. |

### 31.2.4.14 static bool WDOG\_HAL\_GetWorkInDebugModeCmd ( *uint32\_t baseAddr* ) [**inline**], [**static**]

This function checks whether the WDOG works in the CPU debug mode.

Parameters

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

Returns

false means not work while in CPU debug mode, true means works while in CPU debug mode.

### 31.2.4.15 static void WDOG\_HAL\_SetWorkInStopModeCmd ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function configures whether the WDOG is enabled in the CPU stop mode. Make sure that the WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                 |                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | The WDOG peripheral base address                                                                    |
| <i>enable</i>   | false means watchdog is disabled in CPU stop mode. true means watchdog is enabled in CPU stop mode. |

### 31.2.4.16 static bool WDOG\_HAL\_GetWorkInStopModeCmd ( *uint32\_t baseAddr* ) [inline], [static]

This function checks whether the WDOG works in the CPU stop mode. Make sure WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the WDOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

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

Returns

false means not work while in CPU stop mode, true means works while in CPU stop mode.

### 31.2.4.17 static void WDOG\_HAL\_SetWorkInWaitModeCmd ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function configures whether the WDOG is enabled in the CPU wait mode. Make sure WDOG registers are unlocked by the WDOG\_HAL\_Unlock, that the WCT window is still open and that the W-

## WDOG HAL driver

DOG\_STCTRLH register has not been written in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                 |                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | The WDOG peripheral base address                                                                    |
| <i>enable</i>   | false means watchdog is disabled in CPU wait mode. true means watchdog is enabled in CPU wait mode. |

### 31.2.4.18 static bool WDOG\_HAL\_GetWorkInWaitModeCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function checks whether the WDOG works in the CPU wait mode.

Parameters

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

Returns

false means not work while in CPU wait mode, true means works while in CPU wait mode.

### 31.2.4.19 static bool WDOG\_HAL\_IsIntPending ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the WDOG interrupt flag.

Parameters

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

Returns

Watchdog interrupt status, false means interrupt not asserted, true means interrupt asserted.

### 31.2.4.20 static void WDOG\_HAL\_ClearIntFlag ( uint32\_t *baseAddr* ) [inline], [static]

This function clears the WDOG interrupt flag.

Parameters

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

### 31.2.4.21 static void WDOG\_HAL\_SetTimeoutValue ( uint32\_t *baseAddr*, uint32\_t *timeoutCount* ) [inline], [static]

This function sets the WDOG\_TOVAL value. It should be ensured that the time-out value for the Watch-dog is always greater than 2xWCT time + 20 bus clock cycles. Make sure WDOG registers are unlocked by the WDOG\_HAL\_Unlock , that the WCT window is still open and that this API has not been called in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                     |                                                       |
|---------------------|-------------------------------------------------------|
| <i>baseAddr</i>     | The WDOG peripheral base address                      |
| <i>timeoutCount</i> | watchdog timeout value, count of watchdog clock tick. |

### 31.2.4.22 static uint32\_t WDOG\_HAL\_GetTimeoutValue ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the WDOG\_TOVAL value.

Parameters

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

Returns

value of register WDOG\_TOVAL.

### 31.2.4.23 static uint32\_t WDOG\_HAL\_GetTimerOutputValue ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the WDOG\_TMROUT value.

Parameters

## WDOG HAL driver

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

Returns

Current value of watchdog timer counter.

### 31.2.4.24 static void WDOG\_HAL\_SetClockPrescalerValueMode ( *uint32\_t baseAddr*, *wdog\_clock\_prescaler\_value\_t clockPrescaler* ) [inline], [static]

This function sets the WDOG clock prescaler. Make sure WDOG registers are unlocked by the WDO-G\_HAL\_Unlock , that the WCT window is still open and that this API has not been called in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                       |                                                                              |
|-----------------------|------------------------------------------------------------------------------|
| <i>baseAddr</i>       | The WDOG peripheral base address                                             |
| <i>clockPrescaler</i> | watchdog clock prescaler, see <a href="#">wdog_clock_prescaler_value_t</a> . |

### 31.2.4.25 static *wdog\_clock\_prescaler\_value\_t* WDOG\_HAL\_GetClockPrescalerValueMode ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the WDOG clock prescaler.

Parameters

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

Returns

WDOG clock prescaler, see [wdog\\_clock\\_prescaler\\_value\\_t](#).

### 31.2.4.26 static void WDOG\_HAL\_SetWindowValue ( *uint32\_t baseAddr*, *uint32\_t windowValue* ) [inline], [static]

This function sets the WDOG\_WIN value. Make sure WDOG registers are unlocked by the WDOG\_HA-L\_Unlock , that the WCT window is still open and that this API has not been called in this WCT while this function is called. Make sure WDOG\_STCTRLH.ALLOWUPDATE is 1 which means register update is enabled.

Parameters

|                    |                                  |
|--------------------|----------------------------------|
| <i>baseAddr</i>    | The WDOG peripheral base address |
| <i>windowValue</i> | watchdog window value.           |

### 31.2.4.27 static uint32\_t WDOG\_HAL\_GetWindowValue ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the WDOG\_WIN value.

Parameters

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

Returns

watchdog window value.

### 31.2.4.28 static void WDOG\_HAL\_Unlock ( uint32\_t *baseAddr* ) [inline], [static]

This function unlocks the WDOG register written. This function must be called before any configuration is set because watchdog register will be locked automatically after a WCT(256 bus cycles).

Parameters

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

### 31.2.4.29 static void WDOG\_HAL\_Refresh ( uint32\_t *baseAddr* ) [inline], [static]

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

Parameters

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

### 31.2.4.30 static void WDOG\_HAL\_ResetSystem ( uint32\_t *baseAddr* ) [inline], [static]

This function resets the chip using WDOG.

## WDOG HAL driver

Parameters

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

**31.2.4.31 static uint32\_t WDOG\_HAL\_GetResetCount( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the value of the WDOG\_RSTCNT.

Parameters

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

Returns

Chip reset count that was reset by Watchdog.

**31.2.4.32 static void WDOG\_HAL\_ClearResetCount( uint32\_t *baseAddr* ) [inline], [static]**

This function clears the WDOG\_RSTCNT.

Parameters

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

**31.2.4.33 void WDOG\_HAL\_Init( uint32\_t *baseAddr* )**

This function restores the WDOG module to reset value.

Parameters

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

## 31.3 WDOG Peripheral driver

### 31.3.1 Overview

This section describes the programming interface of the WDOG Peripheral driver. The WDOG driver configures and initializes the WDOG.

### 31.3.2 WDOG Initialization

To initialize the WDOG module, call the [WDOG\\_DRV\\_Init\(\)](#) function and pass in the user configuration structure. This function automatically enables the WDOG module and clock.

After the [WDOG\\_DRV\\_Init\(\)](#) function is called, the WDOG is enabled and its counter is working. Therefore, the [WDOG\\_DRV\\_Refresh\(\)](#) function should be called before the WDOG times out.

This example code shows how to initialize and configure the driver:

```
// Define device configuration.
const wdog_user_config_t init =
{
 .clockSource = kWdogClockSourceLpoClock, // WDOG clock source is LPO clock //
 .clockPrescalerValue = kWdogClockPrescalerValueDevide1, // Clock
 prescaler divide by 1 //
 .timeoutValue = 2048, // Timeout count is 2048 //
 .interruptEnable = false, // Interrupt configure, false means disable interrupt //
 .updateRegisterEnable = true, // Enable WDOG register update after first configure //
 .workInWaitModeEnable = true, // Enable WDOG while CPU is in Wait mode //
 .workInStopModeEnable = true, // Enable WDOG while CPU is in Stop mode //
};

// Initialize WDOG.
WDOG_DRV_Init(&init);
```

### 31.3.3 WDOG Refresh

After the WDOG is enabled, the [WDOG\\_DRV\\_Refresh\(\)](#) function should be called periodically to prevent the WDOG from timing out.

Otherwise, a reset is asserted. This is called "Feed Dog".

### 31.3.4 WDOG Reset Count

The WDOG can record the reset count caused by the WDOG timeout.

1. [WDOG\\_DRV\\_GetResetCount\(\)](#) gets the reset count caused by the WDOG timeout.
2. [WDOG\\_DRV\\_ClearResetCount\(\)](#) clears the reset count caused by the WDOG timeout.

## WDOG Peripheral driver

### 31.3.5 WDOG Reset System

The WDOG can be used to reset the MCU.

1. [WDOG\\_DRV\\_ResetSystem\(\)](#) resets the MCU whether the WDOG is enabled or not.

### 31.3.6 WDOG interrupt

If the WDOG interrupt is enabled, the WDOG asserts an interrupt and resets the system after 256 bus clock.

1. Enable the WDOG interrupt with the `wdog_user_config_t.interruptEnable = true`.
2. Define the WDOG IRQ function.

```
void Watchdog_IRQHandler()
{
 // Enter WDOG ISR //
}
```

## Data Structures

- struct [wdog\\_user\\_config\\_t](#)  
*Data structure for Watchdog initialization. [More...](#)*

## Watchdog Driver

- void [WDOG\\_DRV\\_Init](#) (const [wdog\\_user\\_config\\_t](#) \*userConfigPtr)  
*Initializes the Watchdog.*
- void [WDOG\\_DRV\\_Deinit](#) (void)  
*Shuts down the Watchdog.*
- void [WDOG\\_DRV\\_Refresh](#) (void)  
*Refreshes the Watchdog.*
- uint32\_t [WDOG\\_DRV\\_GetResetCount](#) (void)  
*Gets the MCU reset count that is reset by the Watchdog.*
- void [WDOG\\_DRV\\_ClearResetCount](#) (void)  
*Clears the Watchdog reset count.*
- bool [WDOG\\_DRV\\_IsRunning](#) (void)  
*Gets the Watchdog running status.*
- void [WDOG\\_DRV\\_ResetSystem](#) (void)  
*Resets the MCU by the Watchdog.*

### 31.3.7 Data Structure Documentation

#### 31.3.7.1 struct wdog\_user\_config\_t

This structure is used when initializing the WDOG during the `wdog_init` function call. It contains all WDOG configurations.

#### Data Fields

- `clock_wdog_src_t clockSource`  
*Clock source select.*
- `wdog_clock_prescaler_value_t clockPrescalerValue`  
*Clock prescaler value.*
- `bool updateRegisterEnable`  
*Update write-once register enable.*
- `bool workInDebugModeEnable`  
*Enable watchdog while in debug mode.*
- `bool workInWaitModeEnable`  
*Enable watchdog while in wait mode.*
- `bool workInStopModeEnable`  
*Enable watchdog while in stop mode.*
- `uint32_t windowValue`  
*Window value.*
- `uint32_t timeoutValue`  
*Timeout value.*

### 31.3.8 Function Documentation

#### 31.3.8.1 void WDOG\_DRV\_Init ( const wdog\_user\_config\_t \* userConfigPtr )

This function initializes the WDOG. When called, the WDOG runs according to the requirements of the configuration.

Parameters

|                            |                                                                                  |
|----------------------------|----------------------------------------------------------------------------------|
| <code>userConfigPtr</code> | Watchdog user configure data structure, see <a href="#">wdog_user_config_t</a> . |
|----------------------------|----------------------------------------------------------------------------------|

#### 31.3.8.2 void WDOG\_DRV\_Deinit ( void )

This function shuts down the WDOG.

## **WDOG Peripheral driver**

### **31.3.8.3 void WDOG\_DRV\_Refresh( void )**

This function feeds the WDOG. It sets the WDOG timer count to zero and should be called before the Watchdog timer times out. Otherwise, a reset is asserted. Enough time should be allowed for the refresh sequence to be detected by the Watchdog timer on the Watchdog clock.

### **31.3.8.4 uint32\_t WDOG\_DRV\_GetResetCount( void )**

This function gets the WDOG\_RSTCNT value.

Returns

Chip reset count that is reset by the Watchdog.

### **31.3.8.5 void WDOG\_DRV\_ClearResetCount( void )**

This function sets the WDOG reset count to zero. The WDOG\_RSTCNT register clears either on Power-On-Reset or is cleared by this function.

### **31.3.8.6 bool WDOG\_DRV\_IsRunning( void )**

This function gets the WDOG running status.

Returns

watchdog running status, false means not running, true means running

### **31.3.8.7 void WDOG\_DRV\_ResetSystem( void )**

This function resets the MCU by using the WDOG.

# Chapter 32

## Low-Leakage Wakeup Unit (LLWU)

### 32.1 Overview

The Kinetis SDK provides a HAL driver for the Low-Leakage Wakeup Unit (LLWU) block of Kinetis devices. The LLWU module allows the user to select up to 16 external pin sources and up to 8 internal modules as a wake-up source from low-leakage power modes.

### 32.2 Example

The input sources are described in the device's device configuration details. Each of the available wake-up sources can be individually enabled.

This is an example of LLWU HAL access APIs.

```
#include "llwu/hal/fsl_llwu_hal.h"

// set to enable pin 2 as wakeup source using rising edge //
LLWU_HAL_SetExternalInputPinMode(kLlwuExternalPinRisingEdge, 2);

// set to enable pin 8 as wakeup source using change detected //
LLWU_HAL_SetExternalInputPinMode(kLlwuExternalPinChangeDetect, 8);

// Set to enable internal module 1 as wakeup source //
LLWU_HAL_SetInternalModuleCmd(1, true);
```

## Files

- file [fsl\\_llwu\\_hal.h](#)

## Data Structures

- struct [llwu\\_external\\_pin\\_filter\\_mode\\_t](#)  
*External input pin filter control structure.* [More...](#)
- struct [llwu\\_reset\\_enable\\_mode\\_t](#)  
*Reset enable control structure.* [More...](#)

## Enumerations

- enum [llwu\\_external\\_pin\\_modes\\_t](#)  
*External input pin control modes.*
- enum [llwu\\_filter\\_modes\\_t](#)  
*Digital filter control modes.*

## Data Structure Documentation

### 32.3 Data Structure Documentation

32.3.1 `struct llwu_external_pin_filter_mode_t`

32.3.2 `struct llwu_reset_enable_mode_t`

# Chapter 33

## Multipurpose Clock Generator (MCG)

### 33.1 Overview

The Kinetis SDK provides a HAL driver for the Multipurpose Clock Generator (MCG) block of Kinetis devices.

### Modules

- MCG HAL driver

### 33.2 MCG HAL driver

#### 33.2.1 Overview

The chapter describes the programming interface of the MCG Hal driver. The multipurpose clock generator (MCG) module provides several clock source choices for the MCU. The MCG HAL provides a set of APIs to access these registers, including these services:

- OSC related APIs;
- FLL reference clock source, divider and output frequency;
- PLL reference clock source, divider and output frequency;
- MCG mode related APIs;
- MCG auto trim machine function;

#### 33.2.2 OSC related APIs

MCG uses the OSC as an external reference clock source. To use OSC, RANGE, HGO and EREFS register bit fields must be set correctly according to the board configuration. MCG HAL driver provides separate APIs to set these bit fields and get their values. The [CLOCK\\_HAL\\_SetOsc0Mode\(\)](#) combined function enables setting these bit fields in one step.

The OSC frequency is saved in global variables `g_xtal0ClkFreq`, `g_xtal1ClkFreq` and `g_xtalRtcClkFreq`, which are used to calculate the MCG output frequency. Set these variables according to the board settings.

When there are multiple OSC instances, use the [CLOCK\\_HAL\\_SetOscselMode\(\)](#) function to choose which OSC to use. The [CLOCK\\_HAL\\_TestOscFreq\(\)](#) function tests different OSCs frequencies according to parameters.

#### 33.2.3 FLL reference clock source, divider, and output frequency.

To use MCG FLL, ensure that the FLL reference clock, reference clock divider, and FLL DCO is configured correctly. MCG HAL driver provides separate functions to set these registers and get their values. Besides these basic functions, there are additional functions to ensure that the MCG is easy to use.

MCG FLL uses either the internal clock or the external clock as a reference clock. When selecting the external clock, the [CLOCK\\_HAL\\_TestFllExternalRefFreq\(\)](#) function calculates the PLL external reference clock frequency according to parameters. It is useful to validate the parameters before setting them to registers.

The [CLOCK\\_HAL\\_GetFllRefClk\(\)](#) function returns the current FLL reference clock frequency. Function [CLOCK\\_HAL\\_TestFllFreq\(\)](#) function calculates the FLL output frequency based on the input reference clock frequency and the DCO settings. It can be used to validate the setting values before setting them to registers. The [CLOCK\\_HAL\\_GetFllClk\(\)](#) function returns the current FLL frequency.

### 33.2.4 PLL reference clock source, divider and output frequency

PLL uses OSC as an external reference clock. To use MCG PLL, ensure that the reference clock, PRDIV and VDIV are set correctly. MCG HAL driver provides separate APIs for these register bit fields.

The register bit fields PRDIV and VDIV can be configured to generate the desired output frequency. See the appropriate reference manual for more information about setting values. For easy use, MCG HAL driver provides a function `CLOCK_HAL_CalculatePllDiv()` function. Based on the reference clock frequency and the desired output frequency, this function calculates the available PRDIV and VDIV value to ensure that the output frequency is as aligned as possible to the desired frequency.

### 33.2.5 MCG mode related APIs

MCG HAL driver provides functions for mode check and mode transition.

The `CLOCK_HAL_GetMcgMode()` function gets the current MCG mode according to the MCG internal register.

Mode transition functions are defined as `CLOCK_HAL_SetXXXMode`, where XXX is the target mode name. The key register bit fields such as PRDIV, VDIV, FRDIV and DMX32 should be passed as parameters to set the MCG to a specific state. If these parameters are invalid or can't reach the target mode directly, an error is returned.

If the FLL is the target mode, applications should pass a delay function, which ensures that the FLL is stable after changing the reference clock, FRDIV, DMX32 and DRS. Please check the appropriate datasheet for more information about delay times.

### 33.2.6 MCG auto trim machine(ATM) function

The auto trim machine automatically trims the MCG internal reference clock using an external reference clock. See the appropriate reference manual for more details.

The MCG HAL driver provides separate functions for ATM registers. The `CLOCK_HAL_TrimInternalRefClk()` function trims the IRC clock to a desired frequency. The bus clock, in the range of 8-16 MHz, should be the only reference clock and the IRC should not be the MCGOUTCLK source. Otherwise, the ATM does not work and the function returns an error.

## Files

- file `fsl_mcg_hal.h`

## Enumerations

- enum `_mcg_constant`

## MCG HAL driver

- enum `mcg_clock_select_t`  
*MCG clock source select.*
- enum `mcg_internal_ref_clock_source_t`  
*MCG internal reference clock source select.*
- enum `mcg_freq_range_select_t`  
*MCG frequency range select.*
- enum `mcg_high_gain_osc_select_t`  
*MCG high gain oscillator select.*
- enum `mcg_external_ref_clock_select_t`  
*MCG external reference clock select.*
- enum `mcg_low_power_select_t`  
*MCG low power select.*
- enum `mcg_internal_ref_clock_select_t`  
*MCG internal reference clock select.*
- enum `mcg_dmx32_select_t`  
*MCG DCO Maximum Frequency with 32.768 kHz Reference.*
- enum `mcg_dco_range_select_t`  
*MCG DCO range select.*
- enum `mcg_pll_external_ref_clk_select_t`  
*MCG PLL external reference clock select.*
- enum `mcg_pll_select_t`  
*MCG PLL select.*
- enum `mcg_loss_of_lock_status_t`  
*MCG loss of lock status.*
- enum `mcg_lock_status_t`  
*MCG lock status.*
- enum `mcg_pll_stat_status_t`  
*MCG clock status.*
- enum `mcg_internal_ref_status_t`  
*MCG iref status.*
- enum `mcg_clk_stat_status_t`  
*MCG clock mode status.*
- enum `mcg_internal_ref_clk_status_t`  
*MCG ircst status.*
- enum `mcg_atm_fail_status_t`  
*MCG auto trim fail status.*
- enum `mcg_locs0_status_t`  
*MCG loss of clock status.*
- enum `mcg_atm_select_t`  
*MCG Automatic Trim Machine Select.*
- enum `mcg_oscsel_select_t`  
*MCG OSC Clock Select.*
- enum `mcg_loss_of_clk1_status_t`  
*MCG loss of clock status.*
- enum `mcg_pll_clk_select_t`  
*MCG PLLCS select.*
- enum `mcg_locs2_status_t`  
*MCG loss of clock status.*
- enum `mcg_atm_error_t`  
*MCG auto trim machine error code.*

- enum `mcg_status_t` {
   
  `kStatus_MCG_Success` = 0U,
   
  `kStatus_MCG_Fail` = 1U }
   
    *MCG status.*
- enum `mcg_modes_t`
  
    *MCG mode definitions.*
- enum `mcg_mode_error_t`
  
    *MCG mode transition API error code definitions.*

## Functions

- `mcg_modes_t CLOCK_HAL_GetMcgMode` (`uint32_t` baseAddr)
   
    *Gets the current MCG mode.*
- `mcg_mode_error_t CLOCK_HAL_SetFeiMode` (`uint32_t` baseAddr, `mcg_dco_range_select_t` drs, `void(*fllStableDelay)(void)`, `uint32_t *outClkFreq`)
   
    *Set MCG to FEI mode.*
- `mcg_mode_error_t CLOCK_HAL_SetFeeMode` (`uint32_t` baseAddr, `mcg_oscSEL_select_t` oscselVal, `uint8_t` frdivVal, `mcg_dmx32_select_t` dmx32, `mcg_dco_range_select_t` drs, `void(*fllStableDelay)(void)`, `uint32_t *outClkFreq`)
   
    *Set MCG to FEE mode.*
- `mcg_mode_error_t CLOCK_HAL_SetFbiMode` (`uint32_t` baseAddr, `mcg_dco_range_select_t` drs, `mcg_internal_ref_clock_select_t` ircSelect, `uint8_t` fcrdivVal, `void(*fllStableDelay)(void)`, `uint32_t *outClkFreq`)
   
    *Set MCG to FBI mode.*
- `mcg_mode_error_t CLOCK_HAL_SetFbeMode` (`uint32_t` baseAddr, `mcg_oscSEL_select_t` oscselVal, `uint8_t` frdivVal, `mcg_dmx32_select_t` dmx32, `mcg_dco_range_select_t` drs, `void(*fllStableDelay)(void)`, `uint32_t *outClkFreq`)
   
    *Set MCG to FBE mode.*
- `mcg_mode_error_t CLOCK_HAL_SetBlpiMode` (`uint32_t` baseAddr, `uint8_t` fcrdivVal, `mcg_internal_ref_clock_select_t` ircSelect, `uint32_t *outClkFreq`)
   
    *Set MCG to BLPI mode.*
- `mcg_mode_error_t CLOCK_HAL_SetBlpeMode` (`uint32_t` baseAddr, `mcg_oscSEL_select_t` oscselVal, `uint32_t *outClkFreq`)
   
    *Set MCG to BLPE mode.*
- `mcg_mode_error_t CLOCK_HAL_SetPbeMode` (`uint32_t` baseAddr, `mcg_oscSEL_select_t` oscselVal, `mcg_pll_clk_select_t` pllcsSelect, `uint8_t` prdivVal, `uint8_t` vdivVal, `uint32_t *outClkFreq`)
   
    *Set MCG to PBE mode.*
- `mcg_mode_error_t CLOCK_HAL_SetPeeMode` (`uint32_t` baseAddr, `uint32_t *outClkFreq`)
   
    *Set MCG to PBE mode.*

## MCG out clock access API

- `uint32_t CLOCK_HAL_TestOscFreq` (`uint32_t` baseAddr, `mcg_oscSEL_select_t` oscselVal)
- `uint32_t CLOCK_HAL_TestFllExternalRefFreq` (`uint32_t` baseAddr, `uint32_t` extFreq, `uint8_t` frdivVal, `mcg_freq_range_select_t` range0, `mcg_oscSEL_select_t` oscsel)
   
    *Test FLL external reference frequency based on input parameters.*
- `uint32_t CLOCK_HAL_GetFllRefClk` (`uint32_t` baseAddr)

## MCG HAL driver

- Gets the current MCG FLL clock.
- uint32\_t **CLOCK\_HAL\_TestFllFreq** (uint32\_t baseAddr, uint32\_t fllRef, mcg\_dmx32\_select\_t dmx32, mcg\_dco\_range\_select\_t drs)  
    Calculate the FLL frequency based on input parameters.
- uint32\_t **CLOCK\_HAL\_GetFllClk** (uint32\_t baseAddr)  
    Gets the current MCG FLL clock.
- uint32\_t **CLOCK\_HAL\_GetInternalRefClk** (uint32\_t baseAddr)  
    Gets the current MCG internal reference clock(MCGIRCLK).
- uint32\_t **CLOCK\_HAL\_GetFixedFreqClk** (uint32\_t baseAddr)  
    Gets the current MCG fixed frequency clock(MCGFFCLK).
- uint32\_t **CLOCK\_HAL\_GetOutClk** (uint32\_t baseAddr)  
    Gets the current MCG out clock.

## MCG control register access API

- static void **CLOCK\_HAL\_SetClkSrcMode** (uint32\_t baseAddr, mcg\_clock\_select\_t select)  
    Sets the Clock Source Select.
- static mcg\_clock\_select\_t **CLOCK\_HAL\_GetClkSrcMode** (uint32\_t baseAddr)  
    Gets the Clock Source Select.
- static void **CLOCK\_HAL\_SetFllExternalRefDiv** (uint32\_t baseAddr, uint8\_t setting)  
    Sets the FLL External Reference Divider.
- static uint8\_t **CLOCK\_HAL\_GetFllExternalRefDiv** (uint32\_t baseAddr)  
    Gets the FLL External Reference Divider.
- static void **CLOCK\_HAL\_SetInternalRefSelMode** (uint32\_t baseAddr, mcg\_internal\_ref\_clock\_source\_t select)  
    Sets the Internal Reference Select.
- static mcg\_internal\_ref\_clock\_source\_t **CLOCK\_HAL\_GetInternalRefSelMode** (uint32\_t baseAddr)  
    Gets the Internal Reference Select.
- static void **CLOCK\_HAL\_SetClksFrdivInternalRefSelect** (uint32\_t baseAddr, mcg\_clock\_select\_t clks, uint8\_t frdiv, mcg\_internal\_ref\_clock\_source\_t irefs)  
    Sets the CLKS, FRDIV and IREFS at the same time.
- static void **CLOCK\_HAL\_SetInternalClkCmd** (uint32\_t baseAddr, bool enable)  
    Sets the Enable Internal Reference Clock setting.
- static bool **CLOCK\_HAL\_GetInternalClkCmd** (uint32\_t baseAddr)  
    Gets the enable Internal Reference Clock setting.
- static void **CLOCK\_HAL\_SetInternalRefStopCmd** (uint32\_t baseAddr, bool enable)  
    Sets the Internal Reference Clock Stop Enable setting.
- static bool **CLOCK\_HAL\_GetInternalRefStopCmd** (uint32\_t baseAddr)  
    Gets the Enable Internal Reference Clock setting.
- static void **CLOCK\_HAL\_SetLossOfClkReset0Cmd** (uint32\_t baseAddr, bool enable)  
    Sets the Loss of Clock Reset Enable setting.
- static bool **CLOCK\_HAL\_GetLossOfClkReset0Cmd** (uint32\_t baseAddr)  
    Gets the Loss of Clock Reset Enable setting.
- static void **CLOCK\_HAL\_SetRange0Mode** (uint32\_t baseAddr, mcg\_freq\_range\_select\_t select)  
    Sets the Frequency Range Select.
- static mcg\_freq\_range\_select\_t **CLOCK\_HAL\_GetRange0Mode** (uint32\_t baseAddr)  
    Gets the Frequency Range Select.
- static void **CLOCK\_HAL\_SetHighGainOsc0Mode** (uint32\_t baseAddr, mcg\_high\_gain\_osc\_t select)  
    Sets the High Gain Oscillator Mode.

`select_t select)`

*Sets the High Gain Oscillator Select.*

- static `mcg_high_gain_osc_select_t CLOCK_HAL_GetHighGainOsc0Mode` (uint32\_t baseAddr)  
*Gets the High Gain Oscillator Select.*
- static void `CLOCK_HAL_SetExternalRefSel0Mode` (uint32\_t baseAddr, `mcg_external_ref_clock_select_t` select)  
*Sets the External Reference Select.*
- static  
`mcg_external_ref_clock_select_t CLOCK_HAL_GetExternalRefSel0Mode` (uint32\_t baseAddr)  
*Gets the External Reference Select.*
- static void `CLOCK_HAL_SetLowPowerMode` (uint32\_t baseAddr, `mcg_low_power_select_t` select)  
*Sets the Low Power Select.*
- static `mcg_low_power_select_t CLOCK_HAL_GetLowPowerMode` (uint32\_t baseAddr)  
*Gets the Low Power Select.*
- static void `CLOCK_HAL_SetInternalRefClkSelMode` (uint32\_t baseAddr, `mcg_internal_ref_clock_select_t` select)  
*Sets the Internal Reference Clock Select.*
- static  
`mcg_internal_ref_clock_select_t CLOCK_HAL_GetInternalRefClkSelMode` (uint32\_t baseAddr)  
*Gets the Internal Reference Clock Select.*
- static void `CLOCK_HAL_SetSlowInternalRefClkTrim` (uint32\_t baseAddr, uint8\_t setting)  
*Sets the Slow Internal Reference Clock Trim Setting.*
- static uint8\_t `CLOCK_HAL_GetSlowInternalRefClkTrim` (uint32\_t baseAddr)  
*Gets the Slow Internal Reference Clock Trim Setting.*
- static void `CLOCK_HAL_SetDmx32` (uint32\_t baseAddr, `mcg_dmx32_select_t` setting)  
*Sets the DCO Maximum Frequency with 32.768 kHz Reference.*
- static `mcg_dmx32_select_t CLOCK_HAL_GetDmx32` (uint32\_t baseAddr)  
*Gets the DCO Maximum Frequency with the 32.768 kHz Reference Setting.*
- static void `CLOCK_HAL_SetDcoRangeMode` (uint32\_t baseAddr, `mcg_dco_range_select_t` setting)  
*Sets the DCO Range Select.*
- static `mcg_dco_range_select_t CLOCK_HAL_GetDcoRangeMode` (uint32\_t baseAddr)  
*Gets the DCO Range Select Setting.*
- static void `CLOCK_HAL_SetFastInternalRefClkTrim` (uint32\_t baseAddr, uint8\_t setting)  
*Sets the Fast Internal Reference Clock Trim Setting.*
- static uint8\_t `CLOCK_HAL_GetFastInternalRefClkTrim` (uint32\_t baseAddr)  
*Gets the Fast Internal Reference Clock Trim Setting.*
- static void `CLOCK_HAL_SetSlowInternalRefClkFineTrim` (uint32\_t baseAddr, uint8\_t setting)  
*Sets the Slow Internal Reference Clock Fine Trim Setting.*
- static uint8\_t `CLOCK_HAL_GetSlowInternalRefClkFineTrim` (uint32\_t baseAddr)  
*Gets the Slow Internal Reference Clock Fine Trim Setting.*
- static `mcg_internal_ref_status_t CLOCK_HAL_GetInternalRefStatMode` (uint32\_t baseAddr)  
*Gets the Internal Reference Status.*
- static `mcg_clk_stat_status_t CLOCK_HAL_GetClkStatMode` (uint32\_t baseAddr)  
*Gets the Clock Mode Status.*
- static uint8\_t `CLOCK_HAL_GetOscInit0` (uint32\_t baseAddr)  
*Gets the OSC Initialization Status.*
- static  
`mcg_internal_ref_clk_status_t CLOCK_HAL_GetInternalRefClkStatMode` (uint32\_t baseAddr)

## MCG HAL driver

- static `mcg_atm_fail_status_t CLOCK_HAL_GetAutoTrimMachineFailStatus` (uint32\_t baseAddr)  
*Gets the Internal Reference Clock Status.*
- static `mcg_locs0_status_t CLOCK_HAL_GetLocs0Mode` (uint32\_t baseAddr)  
*Gets the Automatic Trim machine Fail Flag.*
- static `void CLOCK_HAL_SetAutoTrimMachineCmd` (uint32\_t baseAddr, bool enable)  
*Gets the OSC0 Loss of Clock Status.*
- static bool `CLOCK_HAL_GetAutoTrimMachineCmd` (uint32\_t baseAddr)  
*Sets the Automatic Trim Machine Enable Setting.*
- static void `CLOCK_HAL_SetAutoTrimMachineSelMode` (uint32\_t baseAddr, `mcg_atm_select_t` setting)  
*Gets the Automatic Trim Machine Enable Setting.*
- static `mcg_atm_select_t CLOCK_HAL_GetAutoTrimMachineSelMode` (uint32\_t baseAddr)  
*Sets the Automatic Trim Machine Select Setting.*
- static void `CLOCK_HAL_SetFllFilterPreserveCmd` (uint32\_t baseAddr, bool enable)  
*Gets the Automatic Trim Machine Select Setting.*
- static bool `CLOCK_HAL_GetFllFilterPreserveCmd` (uint32\_t baseAddr)  
*Sets the FLL Filter Preserve Enable Setting.*
- static void `CLOCK_HAL_SetFastClkInternalRefDiv` (uint32\_t baseAddr, uint8\_t setting)  
*Gets the FLL Filter Preserve Enable Setting.*
- void `CLOCK_HAL_UpdateFastClkInternalRefDiv` (uint32\_t baseAddr, uint8\_t fcrdiv)  
*Sets the Fast Clock Internal Reference Divider Setting.*
- `mcg_status_t CLOCK_HAL_GetAvailableFrdiv` (`mcg_freq_range_select_t` range0, `mcg_oscsel_select_t` oscsel, uint32\_t inputFreq, uint8\_t \*frdiv)  
*Updates the Fast Clock Internal Reference Divider Setting.*
- static uint8\_t `CLOCK_HAL_GetFastClkInternalRefDiv` (uint32\_t baseAddr)  
*Calculates proper FRDIV setting.*
- void `CLOCK_HAL_SetAutoTrimMachineCmpVal` (uint32\_t baseAddr, uint16\_t value)  
*Gets the Fast Clock Internal Reference Divider Setting.*
- uint16\_t `CLOCK_HAL_GetAutoTrimMachineCompVal` (uint32\_t baseAddr)  
*Sets the ATM Compare Value.*
- void `CLOCK_HAL_SetOsc0Mode` (uint32\_t baseAddr, `mcg_freq_range_select_t` range, `mcg_high_gain_osc_select_t` hgo, `mcg_external_ref_clock_select_t` erefs)  
*Gets the ATM Compare Value.*
- `mcg_atm_error_t CLOCK_HAL_TrimInternalRefClk` (uint32\_t baseAddr, uint32\_t extFreq, uint32\_t desireFreq, uint32\_t \*actualFreq, `mcg_atm_select_t` atms)  
*Set the OSC0 work mode.*
- *Auto trim the internal reference clock.*

### 33.2.7 Enumeration Type Documentation

#### 33.2.7.1 enum mcg\_atm\_error\_t

#### 33.2.7.2 enum mcg\_status\_t

Enumerator

`kStatus_MCG_Success` Success.

`kStatus_MCG_Fail` Execution failed.

### 33.2.8 Function Documentation

33.2.8.1 `uint32_t CLOCK_HAL_TestFLLExternalRefFreq ( uint32_t baseAddr, uint32_t extFreq, uint8_t frdivVal, mcg_freq_range_select_t range0, mcg_oscsel_select_t oscsel )`

This function calculates the MCG FLL external reference clock value in frequency(Hertz) based on the input parameters.

## MCG HAL driver

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance.        |
| <i>extFreq</i>  | External OSC frequency.                       |
| <i>fdivVal</i>  | FLL external reference divider value (FRDIV). |
| <i>range0</i>   | OSC0 frequency range selection.               |
| <i>oscsel</i>   | External OSC selection.                       |

Returns

MCG FLL external reference clock frequency.

### 33.2.8.2 `uint32_t CLOCK_HAL_GetFllRefClk ( uint32_t baseAddr )`

This function returns the fll reference clock frequency based on the current MCG configurations and settings. FLL should be properly configured in order to get the valid value.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

value Frequency value in Hertz of FLL reference clock.

### 33.2.8.3 `uint32_t CLOCK_HAL_TestFllFreq ( uint32_t baseAddr, uint32_t fllRef, mcg_dmx32_select_t dmx32, mcg_dco_range_select_t drs )`

This function calculates the FLL frequency based on input parameters.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
| <i>fllRef</i>   | FLL reference clock frequency.         |
| <i>dmx32</i>    | DCO max 32K setting.                   |

|            |                      |
|------------|----------------------|
| <i>drs</i> | DCO range selection. |
|------------|----------------------|

Returns

value Frequency value in Hertz of the mcgflclk.

### 33.2.8.4 `uint32_t CLOCK_HAL_GetFllClk ( uint32_t baseAddr )`

This function returns the mcgflclk value in frequency(Hertz) based on the current MCG configurations and settings. FLL should be properly configured in order to get the valid value.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

value Frequency value in Hertz of the mcgpllclk.

### 33.2.8.5 `uint32_t CLOCK_HAL_GetInternalRefClk ( uint32_t baseAddr )`

This function returns the mcgirclk value in frequency (Hertz) based on the current MCG configurations and settings. It does not check if the mcgirclk is enabled or not, just calculate and return the value.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

value Frequency value in Hertz of the mcgirclk.

### 33.2.8.6 `uint32_t CLOCK_HAL_GetFixedFreqClk ( uint32_t baseAddr )`

This function get the MCGFFCLK, it is only valid when its frequency is not more than MCGOUTCLK/8. If MCGFFCLK is invalid, this function returns 0.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

value Frequency value in Hertz of MCGFFCLK.

### 33.2.8.7 **uint32\_t CLOCK\_HAL\_GetOutClk ( uint32\_t *baseAddr* )**

This function returns the mcgoutclk value in frequency (Hertz) based on the current MCG configurations and settings. The configuration should be properly done in order to get the valid value.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

value Frequency value in Hertz of mcgoutclk.

### 33.2.8.8 **static void CLOCK\_HAL\_SetClkSrcMode ( uint32\_t *baseAddr*, mcg\_clock\_select\_t *select* ) [inline], [static]**

This function selects the clock source for the MCGOUTCLK.

Parameters

|                 |                                                                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance.                                                                                                                                                                                                                                       |
| <i>select</i>   | Clock source selection <ul style="list-style-type: none"><li>• 00: Output of FLL or PLLCS is selected(depends on PLLS control bit)</li><li>• 01: Internal reference clock is selected.</li><li>• 10: External reference clock is selected.</li><li>• 11: Reserved.</li></ul> |

### 33.2.8.9 **static mcg\_clock\_select\_t CLOCK\_HAL\_GetClkSrcMode ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the select of the clock source for the MCGOUTCLK.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select Clock source selection

### 33.2.8.10 static void CLOCK\_HAL\_SetFLLExternalRefDiv ( *uint32\_t baseAddr, uint8\_t setting* ) [inline], [static]

This function sets the FLL External Reference Divider.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
| <i>setting</i>  | Divider setting                        |

### 33.2.8.11 static uint8\_t CLOCK\_HAL\_GetFLLExternalRefDiv ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the FLL External Reference Divider.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting Divider setting

### 33.2.8.12 static void CLOCK\_HAL\_SetInternalRefSelMode ( *uint32\_t baseAddr, mcg\_internal\_ref\_clock\_source\_t select* ) [inline], [static]

This function selects the reference clock source for the FLL.

Parameters

## MCG HAL driver

|                 |                                                                                                                                                                          |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance.                                                                                                                                   |
| <i>select</i>   | Clock source select <ul style="list-style-type: none"><li>• 0: External reference clock is selected</li><li>• 1: The slow internal reference clock is selected</li></ul> |

### 33.2.8.13 static mcg\_internal\_ref\_clock\_source\_t CLOCK\_HAL\_GetInternalRefSelMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the reference clock source for the FLL.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select Clock source select

### 33.2.8.14 static void CLOCK\_HAL\_SetClksFrdivInternalRefSelect ( uint32\_t *baseAddr*, mcg\_clock\_select\_t *clks*, uint8\_t *frdiv*, mcg\_internal\_ref\_clock\_source\_t *irefs* ) [inline], [static]

This function sets the CLKS, FRDIV, and IREFS settings at the same time in order keep the integrity of the clock switching.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
| <i>clks</i>     | Clock source select                    |
| <i>frdiv</i>    | FLL external reference divider select  |
| <i>irefs</i>    | Internal reference select              |

### 33.2.8.15 static void CLOCK\_HAL\_SetInternalClkCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function enables/disables the internal reference clock to use as the MCGIRCLK.

Parameters

|                 |                                                                                                                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. enable Enable or disable internal reference clock. <ul style="list-style-type: none"> <li>• true: MCGIRCLK active</li> <li>• false: MCGIRCLK inactive</li> </ul> |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.16 static bool CLOCK\_HAL\_GetInternalClkCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the reference clock enable setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

enabled True if the internal reference clock is enabled.

### 33.2.8.17 static void CLOCK\_HAL\_SetInternalRefStopCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function controls whether or not the internal reference clock remains enabled when the MCG enters Stop mode.

Parameters

|                 |                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. enable Enable or disable the internal reference clock stop setting. <ul style="list-style-type: none"> <li>• true: Internal reference clock is enabled in Stop mode if IRCLKEN is set or if MCG is in FEI, FBI, or BLPI modes before entering Stop mode.</li> <li>• false: Internal reference clock is disabled in Stop mode</li> </ul> |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.18 static bool CLOCK\_HAL\_GetInternalRefStopCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Internal Reference Clock Stop Enable setting.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

enabled True if internal reference clock stop is enabled.

### 33.2.8.19 static void CLOCK\_HAL\_SetLossOfClkReset0Cmd ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function determines whether an interrupt or a reset request is made following a loss of the OSC0 external reference clock. The LOCRE0 only has an affect when CME0 is set.

Parameters

|                 |                                                                                                                                                                                                                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. enable Loss of Clock Reset Enable setting <ul style="list-style-type: none"><li>• true: Generate a reset request on a loss of OSC0 external reference clock</li><li>• false: Interrupt request is generated on a loss of OSC0 external reference clock</li></ul> |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.20 static bool CLOCK\_HAL\_GetLossOfClkReset0Cmd ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Loss of Clock Reset Enable setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

enabled True if Loss of Clock Reset is enabled.

### 33.2.8.21 static void CLOCK\_HAL\_SetRange0Mode ( *uint32\_t baseAddr, mcg\_freq\_range\_select\_t select* ) [inline], [static]

This function selects the frequency range for the crystal oscillator or an external clock source. See the Oscillator (OSC) chapter for more details and the device data sheet for the frequency ranges used.

Parameters

|                 |                                                                                                                                                                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. select Frequency Range Select <ul style="list-style-type: none"> <li>• 00: Low frequency range selected for the crystal oscillator</li> <li>• 01: High frequency range selected for the crystal oscillator</li> <li>• 1X: Very high frequency range selected for the crystal oscillator</li> </ul> |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.22 static mcg\_freq\_range\_select\_t CLOCK\_HAL\_GetRange0Mode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Frequency Range Select.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select Frequency Range Select

### 33.2.8.23 static void CLOCK\_HAL\_SetHighGainOsc0Mode ( uint32\_t *baseAddr*, mcg\_high\_gain\_osc\_select\_t *select* ) [inline], [static]

This function controls the crystal oscillator mode of operation. See the Oscillator (OSC) chapter for more details.

Parameters

|                 |                                                                                                                                                                                                                                                           |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. select High Gain Oscillator Select. <ul style="list-style-type: none"> <li>• 0: Configure crystal oscillator for low-power operation</li> <li>• 1: Configure crystal oscillator for high-gain operation</li> </ul> |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.24 static mcg\_high\_gain\_osc\_select\_t CLOCK\_HAL\_GetHighGainOsc0Mode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the High Gain Oscillator Select.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select High Gain Oscillator Select

### 33.2.8.25 static void CLOCK\_HAL\_SetExternalRefSel0Mode ( uint32\_t *baseAddr*, mcg\_external\_ref\_clock\_select\_t *select* ) [inline], [static]

This function selects the source for the external reference clock. See the Oscillator (OSC) chapter for more details.

Parameters

|                 |                                                                                                                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. select External Reference Select <ul style="list-style-type: none"><li>• 0: External reference clock requested</li><li>• 1: Oscillator requested</li></ul> |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.26 static mcg\_external\_ref\_clock\_select\_t CLOCK\_HAL\_GetExternalRefSel0Mode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the External Reference Select.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select External Reference Select

### 33.2.8.27 static void CLOCK\_HAL\_SetLowPowerMode ( uint32\_t *baseAddr*, mcg\_low\_power\_select\_t *select* ) [inline], [static]

This function controls whether the FLL (or PLL) is disabled in the BLPI and the BLPE modes. In the FBE or the PBE modes, setting this bit to 1 transitions the MCG into the BLPE mode; in the FBI mode, setting this bit to 1 transitions the MCG into the BLPI mode. In any other MCG mode, the LP bit has no affect..

Parameters

|                 |                                                                                                                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. select Low Power Select <ul style="list-style-type: none"> <li>• 0: FLL (or PLL) is not disabled in bypass modes</li> <li>• 1: FLL (or PLL) is disabled in bypass modes (lower power)</li> </ul> |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### **33.2.8.28 static mcg\_low\_power\_select\_t CLOCK\_HAL\_GetLowPowerMode ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Low Power Select.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select Low Power Select

### **33.2.8.29 static void CLOCK\_HAL\_SetInternalRefClkSelMode ( uint32\_t *baseAddr*, mcg\_internal\_ref\_clock\_select\_t *select* ) [inline], [static]**

This function selects between the fast or slow internal reference clock source.

Parameters

|                 |                                                                                                                                                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. select Low Power Select <ul style="list-style-type: none"> <li>• 0: Slow internal reference clock selected.</li> <li>• 1: Fast internal reference clock selected.</li> </ul> |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### **33.2.8.30 static mcg\_internal\_ref\_clock\_select\_t CLOCK\_HAL\_GetInternalRefClkSelMode ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Internal Reference Clock Select.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

select Internal Reference Clock Select

### 33.2.8.31 static void CLOCK\_HAL\_SetSlowInternalRefClkTrim ( *uint32\_t baseAddr*, *uint8\_t setting* ) [inline], [static]

This function controls the slow internal reference clock frequency by controlling the slow internal reference clock period. The SCTRIM bits are binary weighted (that is, bit 1 adjusts twice as much as bit 0). Increasing the binary value increases the period, and decreasing the value decreases the period. An additional fine trim bit is available in the C4 register as the SCFTRIM bit. Upon reset, this value is loaded with a factory trim value. If an SCTRIM value stored in non-volatile memory is to be used, it is the user's responsibility to copy that value from the non-volatile memory location to this register.

Parameters

|                 |                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting Slow Internal Reference Clock Trim Setting |
|-----------------|-------------------------------------------------------------------------------------------|

### 33.2.8.32 static uint8\_t CLOCK\_HAL\_GetSlowInternalRefClkTrim ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Slow Internal Reference Clock Trim Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting Slow Internal Reference Clock Trim Setting

### 33.2.8.33 static void CLOCK\_HAL\_SetDmx32 ( *uint32\_t baseAddr*, *mcg\_dmx32\_select\_t setting* ) [inline], [static]

This function controls whether or not the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference.

Parameters

|                 |                                                                                                                                                                                                                                                                                 |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting DCO Maximum Frequency with 32.-768 kHz Reference Setting <ul style="list-style-type: none"> <li>• 0: DCO has a default range of 25%.</li> <li>• 1: DCO is fine-tuned for maximum frequency with 32.768 kHz reference.</li> </ul> |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.34 static mcg\_dmx32\_select\_t CLOCK\_HAL\_GetDmx32 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the DCO Maximum Frequency with 32.768 kHz Reference Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting DCO Maximum Frequency with 32.768 kHz Reference Setting

### 33.2.8.35 static void CLOCK\_HAL\_SetDcoRangeMode ( uint32\_t *baseAddr*, mcg\_dco\_range\_select\_t *setting* ) [inline], [static]

This function selects the frequency range for the FLL output, DCOOUT. When the LP bit is set, the writes to the DRS bits are ignored. The DRST read field indicates the current frequency range for the DCOOUT. The DRST field does not update immediately after a write to the DRS field due to internal synchronization between the clock domains. See the DCO Frequency Range table for more details.

Parameters

|                 |                                                                                                                                                                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting DCO Range Select Setting <ul style="list-style-type: none"> <li>• 00: Low range (reset default).</li> <li>• 01: Mid range.</li> <li>• 10: Mid-high range.</li> <li>• 11: High range.</li> </ul> |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.36 static mcg\_dco\_range\_select\_t CLOCK\_HAL\_GetDcoRangeMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the DCO Range Select Setting.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting DCO Range Select Setting

### 33.2.8.37 static void CLOCK\_HAL\_SetFastInternalRefClkTrim ( *uint32\_t baseAddr*, *uint8\_t setting* ) [inline], [static]

This function controls the fast internal reference clock frequency by controlling the fast internal reference clock period. The FCTRIM bits are binary weighted (that is, bit 1 adjusts twice as much as bit 0). Increasing the binary value increases the period, and decreasing the value decreases the period. If an F-CTRIM[3:0] value stored in non-volatile memory is to be used, it is the user's responsibility to copy that value from the non-volatile memory location to this register.

Parameters

|                 |                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting Fast Internal Reference Clock Trim Setting. |
|-----------------|--------------------------------------------------------------------------------------------|

### 33.2.8.38 static uint8\_t CLOCK\_HAL\_GetFastInternalRefClkTrim ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Fast Internal Reference Clock Trim Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting Fast Internal Reference Clock Trim Setting

### 33.2.8.39 static void CLOCK\_HAL\_SetSlowInternalRefClkFineTrim ( *uint32\_t baseAddr*, *uint8\_t setting* ) [inline], [static]

This function controls the smallest adjustment of the slow internal reference clock frequency. Setting the SCFTRIM increases the period and clearing the SCFTRIM decreases the period by the smallest amount possible. If an SCFTRIM value, stored in non-volatile memory, is to be used, it is the user's responsibility to copy that value from the non-volatile memory location to this bit.

Parameters

|                 |                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting Slow Internal Reference Clock Fine Trim Setting |
|-----------------|------------------------------------------------------------------------------------------------|

### 33.2.8.40 static uint8\_t CLOCK\_HAL\_GetSlowInternalRefClkFineTrim ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Slow Internal Reference Clock Fine Trim Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting Slow Internal Reference Clock Fine Trim Setting

### 33.2.8.41 static mcg\_internal\_ref\_status\_t CLOCK\_HAL\_GetInternalRefStatMode ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Internal Reference Status. This bit indicates the current source for the FLL reference clock. The IREFST bit does not update immediately after a write to the IREFS bit due to internal synchronization between the clock domains.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

status Internal Reference Status

- 0: Source of FLL reference clock is the external reference clock.
- 1: Source of FLL reference clock is the internal reference clock.

### 33.2.8.42 static mcg\_clk\_stat\_status\_t CLOCK\_HAL\_GetClkStatMode ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Clock Mode Status. These bits indicate the current clock mode. The CLKST bits do not update immediately after a write to the CLKS bits due to internal synchronization between clock domains.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

status Clock Mode Status

- 00: Output of the FLL is selected (reset default).
- 01: Internal reference clock is selected.
- 10: External reference clock is selected.
- 11: Output of the PLL is selected.

### 33.2.8.43 static uint8\_t CLOCK\_HAL\_GetOscInit0 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the OSC Initialization Status. This bit, which resets to 0, is set to 1 after the initialization cycles of the crystal oscillator clock have completed. After being set, the bit is cleared to 0 if the OSC is subsequently disabled. See the OSC module's detailed description for more information.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

status OSC Initialization Status

### 33.2.8.44 static mcg\_internal\_ref\_clk\_status\_t CLOCK\_HAL\_GetInternalRefClkStatMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Internal Reference Clock Status. The IRCST bit indicates the current source for the internal reference clock select clock (IRCSCLK). The IRCST bit does not update immediately after a write to the IRCS bit due to the internal synchronization between clock domains. The IRCST bit is only updated if the internal reference clock is enabled, either by the MCG being in a mode that uses the IRC or by setting the C1[IRCLKEN] bit.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

status Internal Reference Clock Status

- 0: Source of internal reference clock is the slow clock (32 kHz IRC).
- 1: Source of internal reference clock is the fast clock (2 MHz IRC).

### **33.2.8.45 static mcg\_atm\_fail\_status\_t CLOCK\_HAL\_GetAutoTrimMachineFailStatus ( uint32\_t baseAddr ) [inline], [static]**

This function gets the Automatic Trim machine Fail Flag. This Fail flag for the Automatic Trim Machine (ATM). This bit asserts when the Automatic Trim Machine is enabled (ATME=1) and a write to the C1, C3, C4, and SC registers is detected or the MCG enters into any Stop mode. A write to ATMF clears the flag.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

flag Automatic Trim machine Fail Flag

- 0: Automatic Trim Machine completed normally.
- 1: Automatic Trim Machine failed.

### **33.2.8.46 static mcg\_locs0\_status\_t CLOCK\_HAL\_GetLocs0Mode ( uint32\_t baseAddr ) [inline], [static]**

This function gets the OSC0 Loss of Clock Status. The LOCS0 indicates when a loss of OSC0 reference clock has occurred. The LOCS0 bit only has an effect when CME0 is set. This bit is cleared by writing a logic 1 to it when set.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

status OSC0 Loss of Clock Status

- 0: Loss of OSC0 has not occurred.
- 1: Loss of OSC0 has occurred.

### **33.2.8.47 static void CLOCK\_HAL\_SetAutoTrimMachineCmd ( uint32\_t baseAddr, bool enable ) [inline], [static]**

This function enables/disables the Auto Trim Machine to start automatically trimming the selected Internal Reference Clock. ATME de-asserts after the Auto Trim Machine has completed trimming all trim bits of the IRCS clock selected by the ATMS bit. Writing to C1, C3, C4, and SC registers or entering Stop mode aborts the auto trim operation and clears this bit.

## MCG HAL driver

Parameters

|                 |                                                                                                                                                                                                                   |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. enable Automatic Trim Machine Enable Setting <ul style="list-style-type: none"><li>• true: Auto Trim Machine enabled</li><li>• false: Auto Trim Machine disabled</li></ul> |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.48 static bool CLOCK\_HAL\_GetAutoTrimMachineCmd ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Automatic Trim Machine Enable Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

enabled True if Automatic Trim Machine is enabled

### 33.2.8.49 static void CLOCK\_HAL\_SetAutoTrimMachineSelMode ( *uint32\_t baseAddr,* *mcg\_atm\_select\_t setting* ) [inline], [static]

This function selects the IRCS clock for Auto Trim Test.

Parameters

|                 |                                                                                                                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting Automatic Trim Machine Select Setting <ul style="list-style-type: none"><li>• 0: 32 kHz Internal Reference Clock selected</li><li>• 1: 4 MHz Internal Reference Clock selected</li></ul> |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.50 static mcg\_atm\_select\_t CLOCK\_HAL\_GetAutoTrimMachineSelMode ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the Automatic Trim Machine Select Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting Automatic Trim Machine Select Setting

### 33.2.8.51 static void CLOCK\_HAL\_SetFllFilterPreserveCmd ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function sets the FLL Filter Preserve Enable. This bit prevents the FLL filter values from resetting allowing the FLL output frequency to remain the same during the clock mode changes where the FLL/-DCO output is still valid. (Note: This requires that the FLL reference frequency remain the same as the value prior to the new clock mode switch. Otherwise, the FLL filter and the frequency values change.)

Parameters

|                 |                                                                                                                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. enable FLL Filter Preserve Enable Setting <ul style="list-style-type: none"> <li>• true: FLL filter and FLL frequency retain their previous values during new clock mode change</li> <li>• false: FLL filter and FLL frequency will reset on changes to correct clock mode</li> </ul> |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 33.2.8.52 static bool CLOCK\_HAL\_GetFllFilterPreserveCmd ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the FLL Filter Preserve Enable Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

enabled True if FLL Filter Preserve is enabled.

### 33.2.8.53 static void CLOCK\_HAL\_SetFastClkInternalRefDiv ( *uint32\_t baseAddr, uint8\_t setting* ) [inline], [static]

This function selects the amount to divide down the fast internal reference clock. The resulting frequency is in the range 31.25 kHz to 4 MHz. (Note: Changing the divider when the Fast IRC is enabled is not supported).

## MCG HAL driver

Parameters

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. setting Fast Clock Internal Reference Divider Setting |
|-----------------|----------------------------------------------------------------------------------------------|

### 33.2.8.54 void CLOCK\_HAL\_UpdateFastClkInternalRefDiv ( *uint32\_t baseAddr, uint8\_t fcrdiv* )

This function sets FCRDIV to a new value. FCRDIV can not be changed when fast internal reference is enabled, this function checks the status, if it is enabled, disable it first, then set FCRDIV, at last reenable it. If you can make sure fast internal reference is not enabled, call [CLOCK\\_HAL\\_SetFastClkInternalRefDiv\(\)](#) will be more effective.

Parameters

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. fcrdiv Fast Clock Internal Reference Divider Setting |
|-----------------|---------------------------------------------------------------------------------------------|

### 33.2.8.55 mcg\_status\_t CLOCK\_HAL\_GetAvailableFrdiv ( *mcg\_freq\_range\_select\_t range0, mcg\_oscsel\_select\_t oscsel, uint32\_t inputFreq, uint8\_t \* frdiv* )

This function calculates the proper FRDIV setting according to FLL reference clock. FLL reference clock frequency after FRDIV must be in the range of 31.25kHz to 39.0625kHz.

Parameters

|                  |                                             |
|------------------|---------------------------------------------|
| <i>range0</i>    | RANGE0 setting.                             |
| <i>oscsel</i>    | OSCSEL setting.                             |
| <i>inputFreq</i> | The reference clock frequency before FRDIV. |
| <i>frdiv</i>     | FRDIV result.                               |

Return values

|                            |                             |
|----------------------------|-----------------------------|
| <i>kStatus_MCG_Success</i> | Proper FRDIV is got.        |
| <i>kStatus_MCG_Fail</i>    | Could not get proper FRDIV. |

### 33.2.8.56 static uint8\_t CLOCK\_HAL\_GetFastClkInternalRefDiv ( *uint32\_t baseAddr* ) [**inline**], [**static**]

This function gets the Fast Clock Internal Reference Divider Setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting Fast Clock Internal Reference Divider Setting

### 33.2.8.57 void CLOCK\_HAL\_SetAutoTrimMachineCmpVal ( *uint32\_t baseAddr, uint16\_t value* )

This function sets the ATM compare value. The values are used by the Auto Trim Machine to compare and adjust the Internal Reference trim values during the ATM SAR conversion.

Parameters

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. value ATM Compare Value. |
|-----------------|-----------------------------------------------------------------|

### 33.2.8.58 uint16\_t CLOCK\_HAL\_GetAutoTrimMachineCompVal ( *uint32\_t baseAddr* )

This function gets the ATM Compare Value.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

setting ATM Compare Value.

### 33.2.8.59 void CLOCK\_HAL\_SetOsc0Mode ( *uint32\_t baseAddr, mcg\_freq\_range\_select\_t range, mcg\_high\_gain\_osc\_select\_t hgo, mcg\_external\_ref\_clock\_select\_t erefs* )

This function set OSC0 work mode, include frequency range select, high gain oscillator select and external reference select.

## MCG HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
| <i>range</i>    | Frequency range select.                |
| <i>hgo</i>      | High gain oscillator select.           |
| <i>eref</i>     | External reference select.             |

**33.2.8.60 mcg\_atm\_error\_t CLOCK\_HAL\_TrimInternalRefClk ( uint32\_t *baseAddr*, uint32\_t *extFreq*, uint32\_t *desireFreq*, uint32\_t \* *actualFreq*, mcg\_atm\_select\_t *atms* )**

This function trims the internal reference clock using external clock, if success, return kMcgAtmErrorNone and actual frequency after trimming is got by parameter actualFreq, if error occurs, error code is returned.

Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>baseAddr</i>   | Base address for current MCG instance.         |
| <i>extFreq</i>    | External clock frequency, should be bus clock. |
| <i>desireFreq</i> | Frequency want to trim to.                     |
| <i>actualFreq</i> | Actual frequency after trim.                   |
| <i>atms</i>       | Trim fast or slow internal reference clock.    |

Returns

Return kMcgAtmErrorNone if success, otherwise return error code.

**33.2.8.61 mcg\_modes\_t CLOCK\_HAL\_GetMcgMode ( uint32\_t *baseAddr* )**

This function checks the MCG registers and determine current MCG mode.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current MCG instance. |
|-----------------|----------------------------------------|

Returns

mcgMode Current MCG mode or error code mcg\_modes\_t

**33.2.8.62 mcg\_mode\_error\_t CLOCK\_HAL\_SetFeiMode ( uint32\_t *baseAddr*,  
mcg\_dco\_range\_select\_t *drs*, void(\*)(void) *fllStableDelay*, uint32\_t \* *outClkFreq* )**

This function sets MCG to FEI mode.

## MCG HAL driver

Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>baseAddr</i>       | Base address for current MCG instance.     |
| <i>drs</i>            | The DCO range selection.                   |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. |
| <i>outClkFreq</i>     | MCGCLKOUT frequency in new mode.           |

Returns

value Error code

**33.2.8.63 mcg\_mode\_error\_t CLOCK\_HAL\_SetFeeMode ( uint32\_t *baseAddr*,  
mcg\_oscsel\_select\_t *oscselVal*, uint8\_t *frdivVal*, mcg\_dmx32\_select\_t *dmx32*,  
mcg\_dco\_range\_select\_t *drs*, void(\*)(void) *fllStableDelay*, uint32\_t \* *outClkFreq* )**

This function sets MCG to FEE mode.

Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>baseAddr</i>       | Base address for current MCG instance.     |
| <i>oscselval</i>      | OSCSEL in FEE mode.                        |
| <i>frdivVal</i>       | FRDIV in FEE mode.                         |
| <i>dmx32</i>          | DMX32 in FEE mode.                         |
| <i>drs</i>            | The DCO range selection.                   |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. |
| <i>outClkFreq</i>     | MCGCLKOUT frequency in new mode.           |

Returns

value Error code

**33.2.8.64 mcg\_mode\_error\_t CLOCK\_HAL\_SetFbiMode ( uint32\_t *baseAddr*,  
mcg\_dco\_range\_select\_t *drs*, mcg\_internal\_ref\_clock\_select\_t *ircSelect*, uint8\_t  
*fcrdivVal*, void(\*)(void) *fllStableDelay*, uint32\_t \* *outClkFreq* )**

This function sets MCG to FBI mode.

Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>baseAddr</i>       | Base address for current MCG instance.     |
| <i>drs</i>            | The DCO range selection.                   |
| <i>ircselect</i>      | The internal reference clock to select.    |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. |
| <i>outClkFreq</i>     | MCGCLKOUT frequency in new mode.           |

Returns

value Error code

**33.2.8.65 mcg\_mode\_error\_t CLOCK\_HAL\_SetFbeMode ( uint32\_t *baseAddr*,  
 mcg\_oscsel\_select\_t *oscselVal*, uint8\_t *frdivVal*, mcg\_dmx32\_select\_t *dmx32*,  
 mcg\_dco\_range\_select\_t *drs*, void(\*)(void) *fllStableDelay*, uint32\_t \* *outClkFreq*  
 )**

This function sets MCG to FBE mode.

Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>baseAddr</i>       | Base address for current MCG instance.     |
| <i>oscselval</i>      | OSCSEL in FEE mode.                        |
| <i>frdivVal</i>       | FRDIV in FEE mode.                         |
| <i>dmx32</i>          | DMX32 in FEE mode.                         |
| <i>drs</i>            | The DCO range selection.                   |
| <i>fllStableDelay</i> | Delay function to make sure FLL is stable. |
| <i>outClkFreq</i>     | MCGCLKOUT frequency in new mode.           |

Returns

value Error code

**33.2.8.66 mcg\_mode\_error\_t CLOCK\_HAL\_SetBlpiMode ( uint32\_t *baseAddr*, uint8\_t  
*fcrdivVal*, mcg\_internal\_ref\_clock\_select\_t *ircSelect*, uint32\_t \* *outClkFreq* )**

This function sets MCG to BLPI mode.

## MCG HAL driver

Parameters

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>baseAddr</i>   | Base address for current MCG instance.  |
| <i>ircselect</i>  | The internal reference clock to select. |
| <i>outClkFreq</i> | MCGCLKOUT frequency in new mode.        |

Returns

value Error code

**33.2.8.67 mcg\_mode\_error\_t CLOCK\_HAL\_SetBlpeMode ( uint32\_t *baseAddr*,  
mcg\_oscsel\_select\_t *oscselVal*, uint32\_t \* *outClkFreq* )**

This function sets MCG to BLPE mode.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <i>baseAddr</i>   | Base address for current MCG instance. |
| <i>oscselval</i>  | OSCSEL in FEE mode.                    |
| <i>outClkFreq</i> | MCGCLKOUT frequency in new mode.       |

Returns

value Error code

**33.2.8.68 mcg\_mode\_error\_t CLOCK\_HAL\_SetPbeMode ( uint32\_t *baseAddr*,  
mcg\_oscsel\_select\_t *oscselVal*, mcg\_pll\_clk\_select\_t *pllcsSelect*, uint8\_t  
*prdivVal*, uint8\_t *vdivVal*, uint32\_t \* *outClkFreq* )**

This function sets MCG to PBE mode.

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>baseAddr</i>  | Base address for current MCG instance. |
| <i>oscselval</i> | OSCSEL in FBE mode.                    |

|                    |                                  |
|--------------------|----------------------------------|
| <i>pllcsselect</i> | PLLCS in PBE mode.               |
| <i>prdivval</i>    | PRDIV in PBE mode.               |
| <i>vdivVal</i>     | VDIV in PBE mode.                |
| <i>outClkFreq</i>  | MCGCLKOUT frequency in new mode. |

Returns

value Error code

### 33.2.8.69 mcg\_mode\_error\_t CLOCK\_HAL\_SetPeeMode ( uint32\_t *baseAddr*, uint32\_t \* *outClkFreq* )

This function sets MCG to PBE mode.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <i>baseAddr</i>   | Base address for current MCG instance. |
| <i>outClkFreq</i> | MCGCLKOUT frequency in new mode.       |

Returns

value Error code

Note

This function only change CLKS to use PLL/FLL output. If the PRDIV/VDIV are different from PBE mode, please setup these settings in PBE mode and wait for stable then switch to PEE mode.



# **Chapter 34**

## **Multipurpose Clock Generator Lite (MCG\_Lite)**

### **34.1 Overview**

The Kinetis SDK provides the HAL drivers for the Multipurpose Clock Generator Lite block of Kinetis devices.

### **Modules**

- [MCG\\_Lite HAL driver](#)

### 34.2 MCG\_Lite HAL driver

#### 34.2.1 Overview

The section describes the programming interface of the MCGLite HAL driver. The multi-purpose clock generator Lite (MCG\_Lite) module provides several clock source choices for the MCU. The MCG\_Lite HAL provides a set of APIs to access these registers.

#### 34.2.2 Get current clock output

The MCG\_Lite HAL provides a set of APIs, which are dedicated to getting the different clock frequency, such as MCGOUTCLK, MCGPCLK, MCGIRCLK and LIRC\_CLK. Ensure that the mode setting registers have been configured properly. Otherwise, the API functions are provided with an invalid value.

This example shows how to get a default system reference clock:

```
#include "fsl_mcglite_hal.h"

// return frequency value for specified clock name
uint32_t frequency = 0;

// get the current system default reference clock frequncy
frequency = CLOCK_HAL_GetOutClk();
```

#### 34.2.3 Clock mode switching

MCG\_Lite HAL provides API functions to switch between different clock modes. Before switching, ensure that the operation is valid. For example, a direct switch between the LIRC8M and the LIRC2M is not allowed. Therefore, before switching to the LIRC8M, ensure that the current mode is not LIRC2M.

## Files

- file [fsl\\_mcglite\\_hal.h](#)

## Enumerations

- enum [\\_mcglite\\_constant](#)  
*MCG\_Lite constant definitions.*
- enum [mcglite\\_mcgoutclk\\_source\\_t](#)  
*MCG\_Lite clock source selection.*
- enum [mcglite\\_lirc\\_select\\_t](#)  
*MCG\_Lite LIRC select.*
- enum [mcglite\\_ext\\_select\\_t](#)  
*MCG\_Lite external clock Select.*
- enum [mcglite\\_lirc\\_div\\_t](#)  
*MCG\_Lite divider factor selection for clock source.*

- enum `mcglite_ext_osc_status_t`  
*MCG\_Lite external oscillator status.*
- enum `mcg_freq_range_select_t`  
*MCG frequency range select.*
- enum `mcg_high_gain_osc_select_t`  
*MCG high gain oscillator select.*
- enum `mcglite_mode_t`  
*MCG\_Lite clock mode definitions.*
- enum `mcglite_mode_error_t`  
*MCG\_Lite mode transition API error code definitions.*

## MCG\_Lite output clock access API

- uint32\_t `CLOCK_HAL_GetLircClk` (uint32\_t baseAddr)  
*Gets the current MCG\_Lite low internal reference clock(2MHz or 8MHz)*
- uint32\_t `CLOCK_HAL_GetLircDiv1Clk` (uint32\_t baseAddr)  
*Gets the current MCG\_Lite LIRC\_DIV1\_CLK frequency.*
- uint32\_t `CLOCK_HAL_GetInternalRefClk` (uint32\_t baseAddr)  
*Gets the current MCGIRCLK frequency.*
- uint32\_t `CLOCK_HAL_GetOutClk` (uint32\_t baseAddr)  
*Gets the current MCGETOUTCLK frequency.*

## MCG\_Lite control register access API

- static void `CLOCK_HAL_SetClkSrcMode` (uint32\_t baseAddr, `mcglite_mcgotclk_source_t` select)  
*Sets the Clock Source Select.*
- static `mcglite_mcgotclk_source_t` `CLOCK_HAL_GetClkSrcMode` (uint32\_t baseAddr)  
*Gets the Clock Source Select.*
- static void `CLOCK_HAL_SetLircSelMode` (uint32\_t baseAddr, `mcglite_lirc_select_t` select)  
*Sets the Low Internal Reference Select.*
- static `mcglite_lirc_select_t` `CLOCK_HAL_GetLircSelMode` (uint32\_t baseAddr)  
*Gets the Low Internal Reference Select.*
- static `mcglite_lirc_div_t` `CLOCK_HAL_GetLircRefDiv` (uint32\_t baseAddr)  
*Gets the low internal reference divider 1.*
- static void `CLOCK_HAL_SetLircRefDiv` (uint32\_t baseAddr, `mcglite_lirc_div_t` setting)  
*Sets the low internal reference divider 1.*
- static void `CLOCK_HAL_SetLircDiv2` (uint32\_t baseAddr, `mcglite_lirc_div_t` setting)  
*Sets the low internal reference divider 2.*
- static `mcglite_lirc_div_t` `CLOCK_HAL_GetLircDiv2` (uint32\_t baseAddr)  
*Gets the low internal reference divider 2.*
- static void `CLOCK_HAL_SetLircCmd` (uint32\_t baseAddr, bool enable)  
*Enables the Low Internal Reference Clock setting.*
- static bool `CLOCK_HAL_GetLircCmd` (uint32\_t baseAddr)  
*Get the Low Internal Reference Clock enable or not.*
- static void `CLOCK_HAL_SetLircStopCmd` (uint32\_t baseAddr, bool enable)  
*Sets the Low Internal Reference Clock disabled or not in STOP mode.*
- static bool `CLOCK_HAL_GetLircStopCmd` (uint32\_t baseAddr)

## MCG\_Lite HAL driver

- Gets the Low Internal Reference Clock disabled or not in STOP mode.
- static void [CLOCK\\_HAL\\_SetHircCmd](#) (uint32\_t baseAddr, bool enable)  
    Enable or disable the High Internal Reference Clock setting.
- static bool [CLOCK\\_HAL\\_GetHircCmd](#) (uint32\_t baseAddr)  
    Gets the High Internal Reference Clock is enabled or not.
- static void [CLOCK\\_HAL\\_SetExtRefSelMode0](#) (uint32\_t baseAddr, mcglite\_ext\_select\_t select)  
    Sets the External Reference Select.
- static mcglite\_ext\_select\_t [CLOCK\\_HAL\\_GetExtRefSelMode0](#) (uint32\_t baseAddr)  
    Gets the External Reference Select.
- static mcglite\_mcgotclk\_source\_t [CLOCK\\_HAL\\_GetClkSrcStat](#) (uint32\_t baseAddr)  
    Gets the Clock Mode Status.
- static mcglite\_ext\_osc\_status\_t [CLOCK\\_HAL\\_GetOscInit0](#) (uint32\_t baseAddr)  
    Gets the OSC Initialization Status.

## MCG\_Lite clock mode API

- mcglite\_mode\_t [CLOCK\\_HAL\\_GetMode](#) (uint32\_t baseAddr)  
    Gets the current MCG\_Lite clock mode.
- mcglite\_mode\_error\_t [CLOCK\\_HAL\\_SetHircMode](#) (uint32\_t baseAddr, uint32\_t \*outClkFreq)  
    Sets the MCG\_Lite to HIRC mode.
- mcglite\_mode\_error\_t [CLOCK\\_HAL\\_SetLircMode](#) (uint32\_t baseAddr, mcglite\_lirc\_select\_t lirc, mcglite\_lirc\_div\_t div1, uint32\_t \*outClkFreq)  
    Sets the MCG\_Lite to LIRC mode.
- mcglite\_mode\_error\_t [CLOCK\\_HAL\\_SetExtMode](#) (uint32\_t baseAddr, uint32\_t \*outClkFreq)  
    Sets the MCG\_Lite to EXT mode.

### 34.2.4 Enumeration Type Documentation

#### 34.2.4.1 enum \_mcglite\_constant

#### 34.2.4.2 enum mcglite\_mcgotclk\_source\_t

#### 34.2.4.3 enum mcglite\_lirc\_select\_t

### 34.2.5 Function Documentation

#### 34.2.5.1 uint32\_t [CLOCK\\_HAL\\_GetLircClk](#) ( uint32\_t *baseAddr* )

This function returns the MCG\_Lite LIRC frequency (Hertz) based on the current MCG\_Lite configurations and settings. Please make sure LIRC has been properly configured to get the valid value.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Frequency value in Hertz of the MCG\_Lite LIRC.

#### 34.2.5.2 `uint32_t CLOCK_HAL_GetLircDiv1Clk ( uint32_t baseAddr )`

This function returns the MCG\_Lite LIRC\_DIV1\_CLK frequency (Hertz) based on the current MCG\_Lite configurations and settings. Please make sure LIRC has been properly configured to get the valid value.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Frequency value in Hertz of the MCG\_Lite LIRC\_DIV1\_CLK.

#### 34.2.5.3 `uint32_t CLOCK_HAL_GetInternalRefClk ( uint32_t baseAddr )`

This function returns the MCGIRCLK frequency (Hertz) based on the current MCG\_Lite configurations and settings. Please make sure LIRC has been properly configured to get the valid value.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Frequency value in Hertz of MCGIRCLK.

#### 34.2.5.4 `uint32_t CLOCK_HAL_GetOutClk ( uint32_t baseAddr )`

This function returns the MCGOUTCLK frequency (Hertz) based on the current MCG\_Lite configurations and settings. The configuration should be properly done in order to get the valid value.

## MCG\_Lite HAL driver

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Frequency value in Hertz of MCGOUTCLK.

### 34.2.5.5 static void CLOCK\_HAL\_SetClkSrcMode ( *uint32\_t baseAddr*, mcglite\_mcgoutclk\_source\_t *select* ) [inline], [static]

This function selects the clock source for MCGOUTCLK.

Parameters

|                 |                                                                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address.                                                                                                                                                                                                              |
| <i>select</i>   | Clock source selection <ul style="list-style-type: none"><li>• 00: HIRC clock is select.</li><li>• 01: LIRC(low Internal reference clock) is selected.</li><li>• 10: External reference clock is selected.</li><li>• 11: Reserved.</li></ul> |

### 34.2.5.6 static mcglite\_mcgoutclk\_source\_t CLOCK\_HAL\_GetClkSrcMode ( *uint32\_t baseAddr* ) [inline], [static]

This function checks the register MCG\_C1 CLKSEL and returns the clock source selection for the MCGOUTCLK.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Clock source selection

### 34.2.5.7 static void CLOCK\_HAL\_SetLircSelMode ( *uint32\_t baseAddr*, mcglite\_lirc\_select\_t *select* ) [inline], [static]

This function sets the LIRC to work at 2MHz or 8MHz.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
| <i>Select</i>   | 2MHz or 8MHz.                   |

#### 34.2.5.8 static mcglite\_lirc\_select\_t CLOCK\_HAL\_GetLircSelMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets current LIRC is working at 2MHz or 8MHz.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Current LIRC run status.

#### 34.2.5.9 static mcglite\_lirc\_div\_t CLOCK\_HAL\_GetLircRefDiv ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the low internal reference divider 1, the register FCRDIV.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Current LIRC divider 1 setting.

#### 34.2.5.10 static void CLOCK\_HAL\_SetLircRefDiv ( uint32\_t *baseAddr*, mcglite\_lirc\_div\_t *setting* ) [inline], [static]

This function sets the low internal reference divider 1, the register FCRDIV.

Parameters

## MCG\_Lite HAL driver

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
| <i>setting</i>  | LIRC divider 1 setting value.   |

### 34.2.5.11 static void CLOCK\_HAL\_SetLircDiv2 ( uint32\_t *baseAddr*, mcglite\_lirc\_div\_t *setting* ) [inline], [static]

This function sets the low internal reference divider 2.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
| <i>setting</i>  | LIRC divider 2 setting value.   |

### 34.2.5.12 static mcglite\_lirc\_div\_t CLOCK\_HAL\_GetLircDiv2 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the low internal reference divider 2 setting.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Current LIRC divider 2 setting.

### 34.2.5.13 static void CLOCK\_HAL\_SetLircCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function enables/disables the low internal reference clock.

Parameters

|                 |                                                                                                                                                                            |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address.                                                                                                                                            |
| <i>enable</i>   | Enable or disable internal reference clock. <ul style="list-style-type: none"><li>• true: MCG_Lite Low IRCLK active</li><li>• false: MCG_Lite Low IRCLK inactive</li></ul> |

**34.2.5.14 static bool CLOCK\_HAL\_GetLircCmd ( uint32\_t *baseAddr* ) [inline],  
[static]**

This function checks the low internal reference is enabled or disabled.

## MCG\_Lite HAL driver

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Return values

|              |                  |
|--------------|------------------|
| <i>true</i>  | LIRC is enabled  |
| <i>false</i> | LIRC is disabled |

### 34.2.5.15 static void CLOCK\_HAL\_SetLircStopCmd ( *uint32\_t baseAddr, bool enable* ) [**inline**], [**static**]

This function controls whether or not the low internal reference clock remains enabled when the MCG\_Lite enters STOP mode.

Parameters

|                 |                                                                                                                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address.                                                                                                                                                                                                                                                           |
| <i>enable</i>   | Enable or disable low internal reference clock stop setting. <ul style="list-style-type: none"><li>• true: Internal reference clock is enabled in stop mode if IRCLKEN is set before entering STOP mode.</li><li>• false: Low internal reference clock is disabled in STOP mode</li></ul> |

### 34.2.5.16 static bool CLOCK\_HAL\_GetLircStopCmd ( *uint32\_t baseAddr* ) [**inline**], [**static**]

This function gets the Low Internal Reference Clock Stop Enable setting.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Return values

|              |                                                 |
|--------------|-------------------------------------------------|
| <i>true</i>  | LIRC is enabled in STOP mode if IRCLKEN is set. |
| <i>false</i> | LIRC is disabled in STOP mode.                  |

**34.2.5.17 static void CLOCK\_HAL\_SetHircCmd ( uint32\_t *baseAddr*, bool *enable* )  
[inline], [static]**

This function enables/disables the internal reference clock for use as MCGPCLK.

## MCG\_Lite HAL driver

Parameters

|                 |                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address.                                                                                                              |
| <i>enable</i>   | Enable or disable HIRC. <ul style="list-style-type: none"><li>• true: MCG_Lite HIRC active</li><li>• false: MCG_Lite HIRC inactive</li></ul> |

### 34.2.5.18 static bool CLOCK\_HAL\_GetHircCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the high internal reference clock enable setting.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

True if high internal reference clock is enabled

### 34.2.5.19 static void CLOCK\_HAL\_SetExtRefSelMode0 ( uint32\_t *baseAddr*, mcglite\_ext\_select\_t *select* ) [inline], [static]

This function selects the source for the external reference clock. Refer to the Oscillator (OSC) for more details.

Parameters

|                 |                                                                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address.                                                                                                               |
| <i>select</i>   | External Reference Select. <ul style="list-style-type: none"><li>• 0: External input clock requested</li><li>• 1: Crystal requested</li></ul> |

### 34.2.5.20 static mcglite\_ext\_select\_t CLOCK\_HAL\_GetExtRefSelMode0 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the External Reference Select.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

External Reference Select.

#### 34.2.5.21 static mcglite\_mcgotclk\_source\_t CLOCK\_HAL\_GetClkSrcStat ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Clock Mode Status. These bits indicate the current clock mode. The CLKST bits do not update immediately after a write to the CLKS bits due to internal synchronization between clock domains.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

status Clock Mode Status

- 00: HIRC clock is select.
- 01: LIRC(low Internal reference clock) is selected.
- 10: External reference clock is selected.
- 11: Reserved.

#### 34.2.5.22 static mcglite\_ext\_osc\_status\_t CLOCK\_HAL\_GetOscInit0 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the OSC Initialization Status OSCINIT0. This bit, which resets to 0, is set to 1 after the initialization cycles of the crystal oscillator clock have completed. After being set, the bit is cleared to 0 if the OSC is subsequently disabled. See the OSC module's detailed description for more information.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

OSC initialization status

### 34.2.5.23 mcglite\_mode\_t CLOCK\_HAL\_GetMode ( uint32\_t *baseAddr* )

This is an internal function that checks the MCG registers and determine the current MCG\_lite mode.

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>baseAddr</i> | MCG_Lite register base address. |
|-----------------|---------------------------------|

Returns

Current MCG\_Lite mode or error code.

#### 34.2.5.24 mcglite\_mode\_error\_t CLOCK\_HAL\_SetHircMode ( uint32\_t *baseAddr*, uint32\_t \* *outClkFreq* )

This is an internal function that changes MCG\_Lite to HRIC mode.

Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>baseAddr</i>   | MCG_Lite register base address.  |
| <i>outclkfreq</i> | MCGOUTCLK frequency in new mode. |

Returns

Error code.

#### 34.2.5.25 mcglite\_mode\_error\_t CLOCK\_HAL\_SetLircMode ( uint32\_t *baseAddr*, mcglite\_lirc\_select\_t *lirc*, mcglite\_lirc\_div\_t *div1*, uint32\_t \* *outClkFreq* )

This is an internal function that changes MCG\_Lite to LIRC mode.

Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>baseAddr</i>   | MCG_Lite register base address.  |
| <i>lirc</i>       | Set to LIRC2M or LIRC8M.         |
| <i>div1</i>       | The FCRDIV setting.              |
| <i>outclkfreq</i> | MCGOUTCLK frequency in new mode. |

Returns

Error code.

#### 34.2.5.26 mcglite\_mode\_error\_t CLOCK\_HAL\_SetExtMode ( uint32\_t *baseAddr*, uint32\_t \* *outClkFreq* )

This is an internal function that changes MCG\_Lite to EXT mode. Before this function, please make sure the OSC or external clock source is ready.

## MCG\_Lite HAL driver

Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>baseAddr</i>   | MCG_Lite register base address.  |
| <i>outclkfreq</i> | MCGOUTCLK frequency in new mode. |

Returns

Error code.

# **Chapter 35**

## **Memory-Mapped Divide and Square Root(MMDVSQ)**

### **35.1 Overview**

The Kinetis SDK provides the HAL drivers for the Memory-Mapped Divide and Square Root block of Kinetis devices.

### **Modules**

- [MMDVSQ HAL driver](#)

## MMDVSQ HAL driver

### 35.2 MMDVSQ HAL driver

#### 35.2.1 Overview

The section describes the programming interface of the MMDVSQ HAL driver. The memory-mapped divide and square root (MMDVSQ) module provides hardware divide and square root operation for the MCU. The MMDVSQ HAL provides a set of APIs to access these registers.

#### 35.2.2 Perform MMDVSQ Divide or square root operation

The MMDVSQ HAL provides a set of API functions to set up mathematical computations for division and square root. Ensure that the mode setting registers are configured properly. Otherwise, the API functions are provided with an invalid value.

This is an example to access MMDVSQ HAL APIs:

```
#include "fsl_mmdvsq_hal.h"
// set divide normal start //
MMDVSQ_HAL_SetDivideFastStart(MMDVSQ_BASE, 1)
// perform square root operation of 2 //
result = MMDVSQ_HAL_Sqrt(MMDVSQ_BASE, 2);
```

## Files

- file [fsl\\_mmdvsq\\_hal.h](#)

## Enumerations

- enum [mmdvsq\\_execution\\_status\\_t](#)  
*MMDVSQ execution status.*
- enum [mmdvsq\\_divide\\_operation\\_select\\_t](#)  
*MMDVSQ divide operation select.*
- enum [mmdvsq\\_divide\\_fast\\_start\\_select\\_t](#)  
*MMDVSQ divide fast start select.*
- enum [mmdvsq\\_divide\\_by\\_zero\\_select\\_t](#)  
*MMDVSQ divide by zero setting.*
- enum [mmdvsq\\_divide\\_by\\_zero\\_status\\_t](#)  
*MMDVSQ divide by zero status.*
- enum [mmdvsq\\_unsigned\\_divide\\_select\\_t](#)  
*MMDVSQ unsigned or signed divide calculation select.*
- enum [mmdvsq\\_remainder\\_calculation\\_select\\_t](#)  
*MMDVSQ remainder or quotient result select.*
- enum [mmdvsq\\_error\\_code\\_t](#)  
*MCG mode transition API error code definitions.*

## MMDVSQ operations access API

- `uint32_t MMDVSQ_HAL_DivUR (uint32_t baseAddr, uint32_t dividend, uint32_t divisor)`  
`perform the current MMDVSQ unsigned divide operation and get remainder`
- `uint32_t MMDVSQ_HAL_DivUQ (uint32_t baseAddr, uint32_t dividend, uint32_t divisor)`  
`perform the current MMDVSQ unsigned divide operation and get quotient`
- `uint32_t MMDVSQ_HAL_DivSR (uint32_t baseAddr, uint32_t dividend, uint32_t divisor)`  
`perform the current MMDVSQ signed divide operation and get remainder`
- `uint32_t MMDVSQ_HAL_DivSQ (uint32_t baseAddr, uint32_t dividend, uint32_t divisor)`  
`perform the current MMDVSQ signed divide operation and get quotient`
- `uint16_t MMDVSQ_HAL_Sqrt (uint32_t baseAddr, uint32_t radicand)`  
`set the current MMDVSQ square root operation`

## MMDVSQ control register access API

- static `mmdvsq_execution_status_t MMDVSQ_HAL_GetExecutionStatus (uint32_t baseAddr)`  
`Get the current MMDVSQ execution status.`
- static bool `MMDVSQ_HAL_GetBusyStatus (uint32_t baseAddr)`  
`Get the current MMDVSQ BUSY status.`
- static void `MMDVSQ_HAL_SetDivideFastStart (uint32_t baseAddr, bool enable)`  
`set the current MMDVSQ divide fast start`
- static bool `MMDVSQ_HAL_GetDivideFastStart (uint32_t baseAddr)`  
`get the current MMDVSQ divide fast start setting`
- static void `MMDVSQ_HAL_SetDivdeByZero (uint32_t baseAddr, bool enable)`  
`set the current MMDVSQ divide by zero detection`
- static bool `MMDVSQ_HAL_GetDivdeByZeroSetting (uint32_t baseAddr)`  
`get the current MMDVSQ divide by zero setting`
- static bool `MMDVSQ_HAL_GetDivdeByZeroStatus (uint32_t baseAddr)`  
`get the current MMDVSQ divide by zero status`
- static void `MMDVSQ_HAL_SetRemainderCalculation (uint32_t baseAddr, bool enable)`  
`set the current MMDVSQ divide remainder calculation`
- static bool `MMDVSQ_HAL_GetRemainderCalculation (uint32_t baseAddr)`  
`get the current MMDVSQ divide remainder calculation`
- static void `MMDVSQ_HAL_SetUnsignedCalculation (uint32_t baseAddr, bool enable)`  
`set the current MMDVSQ unsigned divide calculation`
- static bool `MMDVSQ_HAL_GetUnsignedCalculation (uint32_t baseAddr)`  
`get the current MMDVSQ unsigned divide calculation`
- static `uint32_t MMDVSQ_HAL_GetResult (uint32_t baseAddr)`  
`get the current MMDVSQ operation result`
- static void `MMDVSQ_HAL_SetDividend (uint32_t baseAddr, uint32_t dividend)`  
`set the current MMDVSQ dividend value`
- static `uint32_t MMDVSQ_HAL_GetDividend (uint32_t baseAddr)`  
`get the current MMDVSQ dividend value`
- static void `MMDVSQ_HAL_SetDivisor (uint32_t baseAddr, uint32_t divisor)`  
`set the current MMDVSQ divisor value`
- static `uint32_t MMDVSQ_HAL_GetDivisor (uint32_t baseAddr)`  
`get the current MMDVSQ divisor value`
- static void `MMDVSQ_HAL_SetRadicand (uint32_t baseAddr, uint32_t radicand)`  
`set the current MMDVSQ radicand value`

### 35.2.3 Function Documentation

#### 35.2.3.1 `uint32_t MMDVSQ_HAL_DivUR ( uint32_t baseAddr, uint32_t dividend, uint32_t divisor )`

This function performs the MMDVSQ unsigned divide operation and get remainder. It is in block mode. For non-block mode, other HAL routines can be used.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>dividend</i> | -Dividend value                           |
| <i>divisor</i>  | -Divisor value                            |

Returns

Unsigned divide calculation result in the MMDVSQ\_RES register.

### 35.2.3.2 **uint32\_t MMDVSQ\_HAL\_DivUQ ( uint32\_t *baseAddr*, uint32\_t *dividend*, uint32\_t *divisor* )**

This function performs the MMDVSQ unsigned divide operation and get quotient. It is in block mode. For non-block mode, other HAL routines can be used.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>dividend</i> | -Dividend value                           |
| <i>divisor</i>  | -Divisor value                            |

Returns

Unsigned divide calculation result in the MMDVSQ\_RES register.

### 35.2.3.3 **uint32\_t MMDVSQ\_HAL\_DivSR ( uint32\_t *baseAddr*, uint32\_t *dividend*, uint32\_t *divisor* )**

This function performs the MMDVSQ signed divide operation and get remainder. It is in block mode. For non-block mode, other HAL routines can be used.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>dividend</i> | -Dividend value                           |

## MMDVSQ HAL driver

|                |                |
|----------------|----------------|
| <i>divisor</i> | -Divisor value |
|----------------|----------------|

Returns

Signed divide calculation result in the MMDVSQ\_RES register.

### 35.2.3.4 **uint32\_t MMDVSQ\_HAL\_DivSQ ( uint32\_t *baseAddr*, uint32\_t *dividend*, uint32\_t *divisor* )**

This function performs the MMDVSQ signed divide operation and get quotient. It is in block mode. For non-block mode, other HAL routines can be used.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>dividend</i> | -Dividend value                           |
| <i>divisor</i>  | -Divisor value                            |

Returns

Signed divide calculation result in the MMDVSQ\_RES register.

### 35.2.3.5 **uint16\_t MMDVSQ\_HAL\_Sqrt ( uint32\_t *baseAddr*, uint32\_t *radicand* )**

This function performs the MMDVSQ square root operation and return the sqrt result of given radicant-value. It is in block mode. For non-block mode, other HAL routines can be used.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>radicand</i> | - Radicand value                          |

Returns

Square root calculation result in the MMDVSQ\_RES register.

### 35.2.3.6 **static mmdvsq\_execution\_status\_t MMDVSQ\_HAL\_GetExecutionStatus ( uint32\_t *baseAddr* ) [inline], [static]**

This function checks the current MMDVSQ execution status

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

Current MMDVSQ execution status

### 35.2.3.7 static bool MMDVSQ\_HAL\_GetBusyStatus ( uint32\_t *baseAddr* ) [inline], [static]

This function checks the current MMDVSQ BUSY status

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ is busy or idle

### 35.2.3.8 static void MMDVSQ\_HAL\_SetDivideFastStart ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function sets the MMDVSQ divide fast start.

Parameters

|                 |                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance.                                                                                                                                                    |
| <i>enable</i>   | Enable or disable divide fast start mode. <ul style="list-style-type: none"> <li>• true: enable divide fast start.</li> <li>• false: disable divide fast start, use normal start.</li> </ul> |

### 35.2.3.9 static bool MMDVSQ\_HAL\_GetDivideFastStart ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ divide fast start setting

## MMDVSQ HAL driver

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ divide start is fast start or normal start -true : enable fast start mode. -false : disable fast start, divide works normal start mode.

### 35.2.3.10 static void MMDVSQ\_HAL\_SetDivdeByZero ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function sets the MMDVSQ divide by zero detection

Parameters

|                 |                                                                                                                                                                                |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance.                                                                                                                                      |
| <i>enable</i>   | Enable or disable divide by zero detect. <ul style="list-style-type: none"><li>• true: Enable divide by zero detect.</li><li>• false: Disable divide by zero detect.</li></ul> |

### 35.2.3.11 static bool MMDVSQ\_HAL\_GetDivdeByZeroSetting ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ divide by zero setting

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ divide is non-zero divisor or zero divisor

- true: Enable divide by zero detect.
- false: Disable divide by zero detect.

### 35.2.3.12 static bool MMDVSQ\_HAL\_GetDivdeByZeroStatus ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ divide by zero status

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ divide is non-zero divisor or zero divisor

- true: zero divisor.
- false: non-zero divisor.

### 35.2.3.13 static void MMDVSQ\_HAL\_SetRemainderCalculation ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function sets the MMDVSQ divide remainder calculation

Parameters

|                 |                                                                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance.                                                                                                                                                           |
| <i>enable</i>   | Return quotient or remainder in the MMDVSQ_RES register. <ul style="list-style-type: none"> <li>• true: Return remainder in MMQVSQ_RES.</li> <li>• false: Return quotient in MMQVSQ_RES.</li> </ul> |

### 35.2.3.14 static bool MMDVSQ\_HAL\_GetRemainderCalculation ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the MMDVSQ divide remainder calculation

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ divide remainder calculation is quotient or remainder

- true: return remainder in RES register.
- false: return quotient in RES register.

### 35.2.3.15 static void MMDVSQ\_HAL\_SetUnsignedCalculation ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function sets the MMDVSQ unsigned divide calculation

## MMDVSQ HAL driver

Parameters

|                 |                                                                                                                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance.                                                                                                                                                        |
| <i>enable</i>   | Enable or disable unsigned divide calculation. <ul style="list-style-type: none"><li>• true: Enable unsigned divide calculation.</li><li>• false: Disable unsigned divide calculation.</li></ul> |

### 35.2.3.16 static bool MMDVSQ\_HAL\_GetUnsignedCalculation ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ unsigned divide calculation

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ divide is unsigned divide operation

- true: perform an unsigned divide.
- false: perform a signed divide

### 35.2.3.17 static uint32\_t MMDVSQ\_HAL\_GetResult ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ operation result

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ operation result

### 35.2.3.18 static void MMDVSQ\_HAL\_SetDividend ( uint32\_t *baseAddr*, uint32\_t *dividend* ) [inline], [static]

This function sets the MMDVSQ dividend value

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>dividend</i> | Dividend value for divide calculations.   |

### 35.2.3.19 static uint32\_t MMDVSQ\_HAL\_GetDividend ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ dividend value

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ dividend value

### 35.2.3.20 static void MMDVSQ\_HAL\_SetDivisor ( uint32\_t *baseAddr*, uint32\_t *divisor* ) [inline], [static]

This function sets the MMDVSQ divisor value

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>divisor</i>  | Divisor value for divide calculations..   |

### 35.2.3.21 static uint32\_t MMDVSQ\_HAL\_GetDivisor ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the MMDVSQ divisor value

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
|-----------------|-------------------------------------------|

Returns

MMDVSQ divisor value

## MMDVSQ HAL driver

**35.2.3.22 static void MMDVSQ\_HAL\_SetRadicand( uint32\_t baseAddr, uint32\_t radicand ) [inline], [static]**

This function sets the MMDVSQ radicand value

## Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Base address for current MMDVSQ instance. |
| <i>radicand</i> | Radicand value of Sqrt.                   |



# **Chapter 36**

## **Oscillator (OSC)**

### **36.1 Overview**

The Kinetis SDK provides a HAL driver for the Oscillator (OSC) block of Kinetis devices.

### **Modules**

- OSC HAL driver
- Shared OSC Types

## OSC HAL driver

### 36.2 OSC HAL driver

#### 36.2.1 Overview

The section describes the programming interface of the OSC HAL driver. The OSC module is a crystal oscillator. The module and the external crystal or resonator generate a reference clock for the MCU. The osc\_hal provides a set of API functions used to access the SIM registers including enable/disable external reference clock and set the capacitor configurations.

#### 36.2.2 Oscillator Control Register Access API Functions

Each Oscillator has only one control register, OSCx\_CR. It provides an external reference clock enable, external reference clock stop enable, and the capacitor load configuration settings.

Example of OSC HAL access APIs

```
#include "osc/hal/fsl_osc_hal.h"

// Enable the external reference clock
OSC_HAL_SetExternalRefClkCmd(kOsc0, true);

// Disable the external reference clock
OSC_HAL_SetExternalRefClkCmd(kOsc0, false);

#include "osc/hal/fsl_osc_hal.h"

// Enable the osc0 capacitor 2p
OSC_HAL_SetCapacitorCmd(kOsc0, kOscCapacitor2p, true);

// Disable the osc0 capacitor 16p
OSC_HAL_SetCapacitorCmd(kOsc0, kOscCapacitor16p, false);
```

## Files

- file [fsl\\_osc\\_hal.h](#)

## Enumerations

- enum [osc\\_capacitor\\_config\\_t](#) {  
 kOscCapacitor2p = OSC\_CR\_SC2P\_MASK,  
 kOscCapacitor4p = OSC\_CR\_SC4P\_MASK,  
 kOscCapacitor8p = OSC\_CR\_SC8P\_MASK,  
 kOscCapacitor16p = OSC\_CR\_SC16P\_MASK }  
*Oscillator capacitor load configurations.*

## oscillator control APIs

- void [OSC\\_HAL\\_SetExternalRefClkCmd](#) (uint32\_t baseAddr, bool enable)

- bool **OSC\_HAL\_GetExternalRefClkCmd** (uint32\_t baseAddr)
 

*Enables the external reference clock for the oscillator.*
- Gets the external reference clock enable setting for the oscillator.
- void **OSC\_HAL\_SetExternalRefClkInStopModeCmd** (uint32\_t baseAddr, bool enable)
 

*Enables/disables the external reference clock in stop mode.*
- bool **OSC\_HAL\_GetExternalRefClkInStopModeCmd** (uint32\_t baseAddr)
 

*Gets the external reference clock enable setting in stop mode.*
- void **OSC\_HAL\_SetCapacitorCmd** (uint32\_t baseAddr, **osc\_capacitor\_config\_t** capacitorConfig, bool enable)
 

*Enables the capacitor configuration for the oscillator.*
- bool **OSC\_HAL\_GetCapacitorCmd** (uint32\_t baseAddr, **osc\_capacitor\_config\_t** capacitorConfig)
 

*Gets the capacitor configuration for a specific oscillator.*

### 36.2.3 Enumeration Type Documentation

#### 36.2.3.1 enum **osc\_capacitor\_config\_t**

Enumerator

- kOscCapacitor2p* 2 pF capacitor load
- kOscCapacitor4p* 4 pF capacitor load
- kOscCapacitor8p* 8 pF capacitor load
- kOscCapacitor16p* 16 pF capacitor load

### 36.2.4 Function Documentation

#### 36.2.4.1 void **OSC\_HAL\_SetExternalRefClkCmd** ( uint32\_t *baseAddr*, bool *enable* )

This function enables the external reference clock output for the oscillator, OSCERCLK. This clock is used by many peripherals. It should be enabled at an early system initialization stage to ensure the peripherals can select and use it.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | Oscillator register base address |
| <i>enable</i>   | enable/disable the clock         |

#### 36.2.4.2 bool **OSC\_HAL\_GetExternalRefClkCmd** ( uint32\_t *baseAddr* )

This function gets the external reference clock output enable setting for the oscillator , OSCERCLK. This clock is used by many peripherals. It should be enabled at an early system initialization stage to ensure the peripherals could select and use it.

## OSC HAL driver

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | Oscillator register base address |
|-----------------|----------------------------------|

Returns

enable clock enable/disable setting

### **36.2.4.3 void OSC\_HAL\_SetExternalRefClkInStopModeCmd ( uint32\_t *baseAddr*, bool *enable* )**

This function enables/disables the external reference clock (OSCERCLK) when an MCU enters the stop mode.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | Oscillator register base address |
| <i>enable</i>   | enable/disable setting           |

### **36.2.4.4 bool OSC\_HAL\_GetExternalRefClkInStopModeCmd ( uint32\_t *baseAddr* )**

This function gets the external reference clock (OSCERCLK) enable setting when an MCU enters stop mode.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | Oscillator register base address |
|-----------------|----------------------------------|

### **36.2.4.5 void OSC\_HAL\_SetCapacitorCmd ( uint32\_t *baseAddr*, osc\_capacitor\_config\_t *capacitorConfig*, bool *enable* )**

This function enables the specified capacitors configuration for the oscillator. This should be done in the early system level initialization function call based on the system configuration.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | Oscillator register base address |
|-----------------|----------------------------------|

|                         |                                            |
|-------------------------|--------------------------------------------|
| <i>capacitor-Config</i> | Capacitor configuration. (2p, 4p, 8p, 16p) |
| <i>enable</i>           | enable/disable the Capacitor configuration |

### 36.2.4.6 **bool OSC\_HAL\_GetCapacitorCmd ( uint32\_t *baseAddr*, osc\_capacitor\_config\_t *capacitorConfig* )**

This function gets the specified capacitors configuration for an oscillator.

Parameters

|                         |                                  |
|-------------------------|----------------------------------|
| <i>baseAddr</i>         | Oscillator register base address |
| <i>capacitor-Config</i> | Capacitor configuration.         |

Returns

enable enable/disable setting

## Shared OSC Types

### 36.3 Shared OSC Types

The chapter describes the programming interface for the OSC HAL driver.

# Chapter 37

## Power Management Controller (PMC)

### 37.1 Overview

The Kinetis SDK provides a HAL driver for the Power Management Controller (PMC) block of Kinetis devices.

### Modules

- [OSC HAL driver](#)

### 37.2 OSC HAL driver

#### 37.2.1 Overview

The Power Management Controller (PMC) contains the internal voltage regulator, power on reset (POR), and low voltage detect system. The pmc\_hal provides a set of APIs used to access the control registers including enable/disable features provided by this module.

#### 37.2.2 Power Management Controller feature access API functions

1. Internal voltage regulator
2. Active POR providing brown-out detect
3. Low-voltage detect supporting two low-voltage trip points with four warning levels per trip point

This is an example of PMC HAL access API functions.

```
#include "pmc/hal/fsl_pmc_hal.h"

// Low-Voltage Detect Interrupt Enable
PMC_HAL_SetLowVoltIntCmd(kPmcIntLowVoltDetect, true);

// Low-Voltage Detect Interrupt Disable (use polling)
PMC_HAL_SetLowVoltIntCmd(kPmcIntLowVoltDetect, false);

// Set Low-Voltage Detect Voltage Select to Low trip point (V LVD = V LVDL)
PMC_HAL_SetLowVoltDetectVoltMode(
 kPmcLowVoltDetectVoltLowTrip);
```

## Files

- file [fsl\\_pmc\\_hal.h](#)

## Enumerations

- enum [pmc\\_low\\_volt\\_warn\\_volt\\_select\\_t](#) {  
 kPmcLowVoltWarnVoltLowTrip,  
 kPmcLowVoltWarnVoltMid1Trip,  
 kPmcLowVoltWarnVoltMid2Trip,  
 kPmcLowVoltWarnVoltHighTrip }  
*Low-Voltage Warning Voltage Select.*
- enum [pmc\\_low\\_volt\\_detect\\_volt\\_select\\_t](#) {  
 kPmcLowVoltDetectVoltLowTrip,  
 kPmcLowVoltDetectVoltHighTrip }  
*Low-Voltage Detect Voltage Select.*
- enum [pmc\\_int\\_select\\_t](#) {  
 kPmcIntLowVoltDetect,  
 kPmcIntLowVoltWarn }  
*interrupt control*

## Power Management Controller Control APIs

- void [PMC\\_HAL\\_SetLowVoltIntCmd](#) (uint32\_t baseAddr, [pmc\\_int\\_select\\_t](#) intSelect, bool enable)  
*Enables/Disables low voltage-related interrupts.*
- static void [PMC\\_HAL\\_SetLowVoltDetectResetCmd](#) (uint32\_t baseAddr, bool enable)  
*Low-Voltage Detect Hardware Reset Enable/Disable (write once)*
- static void [PMC\\_HAL\\_SetLowVoltDetectAck](#) (uint32\_t baseAddr)  
*Low-Voltage Detect Acknowledge.*
- static bool [PMC\\_HAL\\_GetLowVoltDetectFlag](#) (uint32\_t baseAddr)  
*Low-Voltage Detect Flag Read.*
- static void [PMC\\_HAL\\_SetLowVoltDetectVoltMode](#) (uint32\_t baseAddr, [pmc\\_low\\_volt\\_detect\\_volt\\_select\\_t](#) select)  
*Sets the Low-Voltage Detect Voltage Mode.*
- static [pmc\\_low\\_volt\\_detect\\_volt\\_select\\_t](#) [PMC\\_HAL\\_GetLowVoltDetectVoltMode](#) (uint32\_t baseAddr)  
*Gets the Low-Voltage Detect Voltage Mode.*
- static void [PMC\\_HAL\\_SetLowVoltWarnAck](#) (uint32\_t baseAddr)  
*Low-Voltage Warning Acknowledge.*
- static bool [PMC\\_HAL\\_GetLowVoltWarnFlag](#) (uint32\_t baseAddr)  
*Low-Voltage Warning Flag Read.*
- static void [PMC\\_HAL\\_SetLowVoltWarnVoltMode](#) (uint32\_t baseAddr, [pmc\\_low\\_volt\\_warn\\_volt\\_select\\_t](#) select)  
*Sets the Low-Voltage Warning Voltage Mode.*
- static [pmc\\_low\\_volt\\_warn\\_volt\\_select\\_t](#) [PMC\\_HAL\\_GetLowVoltWarnVoltMode](#) (uint32\_t baseAddr)  
*Gets the Low-Voltage Warning Voltage Mode.*
- static void [PMC\\_HAL\\_SetBandgapBufferCmd](#) (uint32\_t baseAddr, bool enable)  
*Enables/Disables the Bandgap Buffer.*
- static uint8\_t [PMC\\_HAL\\_GetAckIsolation](#) (uint32\_t baseAddr)  
*Gets the acknowledge isolation value.*
- static void [PMC\\_HAL\\_SetClearAckIsolation](#) (uint32\_t baseAddr)  
*Clears an acknowledge isolation.*
- static uint8\_t [PMC\\_HAL\\_GetRegulatorStatus](#) (uint32\_t baseAddr)  
*Gets the Regulator regulation status.*

### 37.2.3 Enumeration Type Documentation

#### 37.2.3.1 enum pmc\_low\_volt\_warn\_volt\_select\_t

Enumerator

- kPmcLowVoltWarnVoltLowTrip** Low trip point selected (VLVW = VLVW1)
- kPmcLowVoltWarnVoltMid1Trip** Mid 1 trip point selected (VLVW = VLVW2)
- kPmcLowVoltWarnVoltMid2Trip** Mid 2 trip point selected (VLVW = VLVW3)
- kPmcLowVoltWarnVoltHighTrip** High trip point selected (VLVW = VLVW4)

## OSC HAL driver

### 37.2.3.2 enum pmc\_low\_volt\_detect\_volt\_select\_t

Enumerator

*kPmcLowVoltDetectVoltLowTrip* Low trip point selected (V LVD = V LVDL )

*kPmcLowVoltDetectVoltHighTrip* High trip point selected (V LVD = V LVDH )

### 37.2.3.3 enum pmc\_int\_select\_t

Enumerator

*kPmcIntLowVoltDetect* Low Voltage Detect Interrupt.

*kPmcIntLowVoltWarn* Low Voltage Warning Interrupt.

## 37.2.4 Function Documentation

### 37.2.4.1 void PMC\_HAL\_SetLowVoltIntCmd ( uint32\_t baseAddr, pmc\_int\_select\_t intSelect, bool enable )

This function enables the interrupt for the low voltage detection, warning, etc. When enabled, if the LVDF (Low Voltage Detect Flag) is set, a hardware interrupt occurs.

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>baseAddr</i>  | Base address for current PMC instance. |
| <i>intSelect</i> | interrupt select                       |
| <i>enable</i>    | enable/disable the interrupt           |

### 37.2.4.2 static void PMC\_HAL\_SetLowVoltDetectResetCmd ( uint32\_t baseAddr, bool enable ) [inline], [static]

This function enables/disables the hardware reset for the low voltage detection. When enabled, if the L-VDF (Low Voltage Detect Flag) is set, a hardware reset occurs. This setting is a write-once-only. Any additional writes are ignored.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
| <i>enable</i>   | enable/disable the LVD hardware reset  |

### 37.2.4.3 static void PMC\_HAL\_SetLowVoltDetectAck ( uint32\_t *baseAddr* ) [inline], [static]

This function acknowledges the low voltage detection errors (write 1 to clear LVDF).

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

### 37.2.4.4 static bool PMC\_HAL\_GetLowVoltDetectFlag ( uint32\_t *baseAddr* ) [inline], [static]

This function reads the current LVDF status. If it returns 1, a low voltage event is detected.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

Returns

- status Current low voltage detect flag
  - true: Low-Voltage detected
  - false: Low-Voltage not detected

### 37.2.4.5 static void PMC\_HAL\_SetLowVoltDetectVoltMode ( uint32\_t *baseAddr*, pmc\_low\_volt\_detect\_volt\_select\_t *select* ) [inline], [static]

This function sets the low voltage detect voltage select. It sets the low voltage detect trip point voltage (Vlvd). An application can select either a low-trip or a high-trip point. See a chip reference manual for details.

Parameters

---

## OSC HAL driver

|                 |                                                     |
|-----------------|-----------------------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance.              |
| <i>select</i>   | Voltage select setting defined in pmc_lvdv_select_t |

### 37.2.4.6 static pmc\_low\_volt\_detect\_volt\_select\_t PMC\_HAL\_GetLowVoltDetectVoltMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the low voltage detect voltage select. It gets the low voltage detect trip point voltage (Vlvd). An application can select either a low-trip or a high-trip point. See a chip reference manual for details.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

Returns

select Current voltage select setting

### 37.2.4.7 static void PMC\_HAL\_SetLowVoltWarnAck ( uint32\_t *baseAddr* ) [inline], [static]

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

### 37.2.4.8 static bool PMC\_HAL\_GetLowVoltWarnFlag ( uint32\_t *baseAddr* ) [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

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

Returns

- status Current LVWF status
- true: Low-Voltage Warning Flag is set.
  - false: the Low-Voltage Warning does not happen.

#### **37.2.4.9 static void PMC\_HAL\_SetLowVoltWarnVoltMode ( uint32\_t *baseAddr*, pmc\_low\_volt\_warn\_volt\_select\_t *select* ) [inline], [static]**

This function sets the low voltage warning voltage select. It sets the low voltage warning trip point voltage (Vlvw). An application can select either a low, mid1, mid2 and a high-trip point. See a chip reference manual for details and the pmc\_lvww\_select\_t for supported settings.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
| <i>select</i>   | Low voltage warning select setting     |

#### **37.2.4.10 static pmc\_low\_volt\_warn\_volt\_select\_t PMC\_HAL\_GetLowVoltWarnVoltMode ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the low voltage warning voltage select. It gets the low voltage warning trip point voltage (Vlvw). See the pmc\_lvww\_select\_t for supported settings.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

Returns

- select* Current low voltage warning select setting

#### **37.2.4.11 static void PMC\_HAL\_SetBandgapBufferCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

This function enables/disables the Bandgap buffer.

## OSC HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
| <i>enable</i>   | enable/disable the Bangap Buffer.      |

### 37.2.4.12 static uint8\_t PMC\_HAL\_GetAckIsolation ( uint32\_t *baseAddr* ) [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

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

Returns

value 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.

### 37.2.4.13 static void PMC\_HAL\_SetClearAckIsolation ( uint32\_t *baseAddr* ) [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

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

### 37.2.4.14 static uint8\_t PMC\_HAL\_GetRegulatorStatus ( uint32\_t *baseAddr* ) [inline], [static]

This function returns the regulator to a run regulation status. It provides the current status of the internal voltage regulator.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current PMC instance. |
|-----------------|----------------------------------------|

## Returns

value Regulation status 0 - Regulator is in a stop regulation or in transition to/from it. 1 - Regulator is in a run regulation.



# **Chapter 38**

## **Port Control and Interrupts (PORT)**

### **38.1 Overview**

The Kinetis SDK provides a HAL driver for the Port Control and Interrupts (PORT) block of Kinetis devices.

### **Modules**

- [PORT HAL driver](#)

## POR T HAL driver

### 38.2 POR T HAL driver

#### 38.2.1 Overview

The section describes the programming interface of the POR T HAL driver. Port control and interrupts hardware driver configuration. Use these functions to set port control and external interrupt functions. Most functions can be configured independently for each pin in the 32-bit port and affect the pin regardless of its pin muxing state. To use these functions, pass to the instance number (HW\_PORTA, HW\_PORTB, HW\_PORTC, etc).

## Files

- file `fsl_port_hal.h`

*The port features such as "digital filter", "pull", etc will be valid when it's available in one of the pins.*

## Enumerations

- enum `port_pull_t` {  
    `kPortPullDown` = 0U,  
    `kPortPullUp` = 1U }  
*Internal resistor pull feature selection.*
- enum `port_slew_rate_t` {  
    `kPortFastSlewRate` = 0U,  
    `kPortSlowSlewRate` = 1U }  
*Slew rate selection.*
- enum `port_drive_strength_t` {  
    `kPortLowDriveStrength` = 0U,  
    `kPortHighDriveStrength` = 1U }  
*Configures the drive strength.*
- enum `port_mux_t` {  
    `kPortPinDisabled` = 0U,  
    `kPortMuxAsGpio` = 1U,  
    `kPortMuxAlt2` = 2U,  
    `kPortMuxAlt3` = 3U,  
    `kPortMuxAlt4` = 4U,  
    `kPortMuxAlt5` = 5U,  
    `kPortMuxAlt6` = 6U,  
    `kPortMuxAlt7` = 7U }  
*Pin mux selection.*
- enum `port_interrupt_config_t` {  
    `kPortIntDisabled` = 0x0U,  
    `kPortIntLogicZero` = 0x8U,  
    `kPortIntRisingEdge` = 0x9U,  
    `kPortIntFallingEdge` = 0xAU,  
    `kPortIntEitherEdge` = 0xBU,

```
kPortIntLogicOne = 0x CU }
```

*Digital filter clock source selection.*

## Configuration

- static void **PORT\_HAL\_SetMuxMode** (uint32\_t baseAddr, uint32\_t pin, **port\_mux\_t** mux)  
*Configures the pin muxing.*
- void **PORT\_HAL\_SetLowGlobalPinCtrl** (uint32\_t baseAddr, uint16\_t lowPinSelect, uint16\_t config)  
*Configures the low half of the pin control register for the same settings.*
- void **PORT\_HAL\_SetHighGlobalPinCtrl** (uint32\_t baseAddr, uint16\_t highPinSelect, uint16\_t config)  
*Configures the high half of pin control register for the same settings.*

## Interrupt

- static void **PORT\_HAL\_SetPinIntMode** (uint32\_t baseAddr, uint32\_t pin, **port\_interrupt\_config\_t** intConfig)  
*Configures the port pin interrupt/DMA request.*
- static **port\_interrupt\_config\_t** **PORT\_HAL\_GetPinIntMode** (uint32\_t baseAddr, uint32\_t pin)  
*Gets the current port pin interrupt/DMA request configuration.*
- static bool **PORT\_HAL\_IsPinIntPending** (uint32\_t baseAddr, uint32\_t pin)  
*Reads the individual pin-interrupt status flag.*
- static void **PORT\_HAL\_ClearPinIntFlag** (uint32\_t baseAddr, uint32\_t pin)  
*Clears the individual pin-interrupt status flag.*
- static uint32\_t **PORT\_HAL\_GetPortIntFlag** (uint32\_t baseAddr)  
*Reads the entire port interrupt status flag.*
- static void **PORT\_HAL\_ClearPortIntFlag** (uint32\_t baseAddr)  
*Clears the entire port interrupt status flag.*

### 38.2.2 Enumeration Type Documentation

#### 38.2.2.1 enum port\_pull\_t

Enumerator

**kPortPullDown** internal pull-down resistor is enabled.

**kPortPullUp** internal pull-up resistor is enabled.

#### 38.2.2.2 enum port\_slew\_rate\_t

Enumerator

**kPortFastSlewRate** fast slew rate is configured.

## POR T HAL driver

*kPortSlowSlewRate* slow slew rate is configured.

### 38.2.2.3 enum port\_drive\_strength\_t

Enumerator

*kPortLowDriveStrength* low drive strength is configured.

*kPortHighDriveStrength* high drive strength is configured.

### 38.2.2.4 enum port\_mux\_t

Enumerator

*kPortPinDisabled* corresponding pin is disabled as analog.

*kPortMuxAsGpio* corresponding pin is configured as GPIO.

*kPortMuxAlt2* chip-specific

*kPortMuxAlt3* chip-specific

*kPortMuxAlt4* chip-specific

*kPortMuxAlt5* chip-specific

*kPortMuxAlt6* chip-specific

*kPortMuxAlt7* chip-specific

### 38.2.2.5 enum port\_interrupt\_config\_t

Configures the interrupt generation condition.

Enumerator

*kPortIntDisabled* Interrupt/DMA request is disabled.

*kPortIntLogicZero* Interrupt when logic zero.

*kPortIntRisingEdge* Interrupt on rising edge.

*kPortIntFallingEdge* Interrupt on falling edge.

*kPortIntEitherEdge* Interrupt on either edge.

*kPortIntLogicOne* Interrupt when logic one.

## 38.2.3 Function Documentation

### 38.2.3.1 static void PORT\_HAL\_SetMuxMode ( uint32\_t baseAddr, uint32\_t pin, port\_mux\_t mux ) [inline], [static]

Parameters

|                 |                                                                                                                                                                                   |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | port base address                                                                                                                                                                 |
| <i>pin</i>      | port pin number                                                                                                                                                                   |
| <i>mux</i>      | pin muxing slot selection <ul style="list-style-type: none"> <li>• kPinDisabled: Pin disabled.</li> <li>• kMuxAsGpio : Set as GPIO.</li> <li>• others : chip-specific.</li> </ul> |

### 38.2.3.2 void PORT\_HAL\_SetLowGlobalPinCtrl ( *uint32\_t baseAddr, uint16\_t lowPinSelect, uint16\_t config* )

This function operates pin 0 -15 of one specific port.

Parameters

|                     |                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i>     | port base address                                                                                                                                                                                                                                                                                                                 |
| <i>lowPinSelect</i> | update corresponding pin control register or not. For a specific bit: <ul style="list-style-type: none"> <li>• 0: corresponding low half of pin control register won't be updated according to configuration.</li> <li>• 1: corresponding low half of pin control register will be updated according to configuration.</li> </ul> |
| <i>config</i>       | value is written to a low half port control register bits[15:0].                                                                                                                                                                                                                                                                  |

### 38.2.3.3 void PORT\_HAL\_SetHighGlobalPinCtrl ( *uint32\_t baseAddr, uint16\_t highPinSelect, uint16\_t config* )

This function operates pin 16 -31 of one specific port.

Parameters

|                      |                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i>      | port base address                                                                                                                                                                                                                                                                                                                   |
| <i>highPinSelect</i> | update corresponding pin control register or not. For a specific bit: <ul style="list-style-type: none"> <li>• 0: corresponding high half of pin control register won't be updated according to configuration.</li> <li>• 1: corresponding high half of pin control register will be updated according to configuration.</li> </ul> |
| <i>config</i>        | value is written to a high half port control register bits[15:0].                                                                                                                                                                                                                                                                   |

### 38.2.3.4 static void PORT\_HAL\_SetPinIntMode ( uint32\_t baseAddr, uint32\_t pin, port\_interrupt\_config\_t intConfig ) [inline], [static]

Parameters

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i>  | port base address.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <i>pin</i>       | port pin number                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>intConfig</i> | interrupt configuration <ul style="list-style-type: none"><li>• kIntDisabled : Interrupt/DMA request disabled.</li><li>• kDmaRisingEdge : DMA request on rising edge.</li><li>• kDmaFallingEdge: DMA request on falling edge.</li><li>• kDmaEitherEdge : DMA request on either edge.</li><li>• KIntLogicZero : Interrupt when logic zero.</li><li>• KIntRisingEdge : Interrupt on rising edge.</li><li>• KIntFallingEdge: Interrupt on falling edge.</li><li>• KIntEitherEdge : Interrupt on either edge.</li><li>• KIntLogicOne : Interrupt when logic one.</li></ul> |

### 38.2.3.5 static port\_interrupt\_config\_t PORT\_HAL\_GetPinIntMode ( uint32\_t baseAddr, uint32\_t pin ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | port base address |
| <i>pin</i>      | port pin number   |

Returns

interrupt configuration

- kIntDisabled : Interrupt/DMA request disabled.
- kDmaRisingEdge : DMA request on rising edge.
- kDmaFallingEdge: DMA request on falling edge.
- kDmaEitherEdge : DMA request on either edge.
- KIntLogicZero : Interrupt when logic zero.
- KIntRisingEdge : Interrupt on rising edge.
- KIntFallingEdge: Interrupt on falling edge.
- KIntEitherEdge : Interrupt on either edge.
- KIntLogicOne : Interrupt when logic one.

### 38.2.3.6 static bool PORT\_HAL\_IsPinIntPending ( *uint32\_t baseAddr, uint32\_t pin* ) [inline], [static]

If a pin is configured to generate the DMA request, the corresponding flag is cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to that flag. If configured for a level sensitive interrupt that remains asserted, the flag is set again immediately.

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | port base address |
| <i>pin</i>      | port pin number   |

Returns

current pin interrupt status flag  
• 0: interrupt is not detected.  
• 1: interrupt is detected.

### 38.2.3.7 static void PORT\_HAL\_ClearPinIntFlag ( *uint32\_t baseAddr, uint32\_t pin* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | port base address |
| <i>pin</i>      | port pin number   |

### 38.2.3.8 static uint32\_t PORT\_HAL\_GetPortIntFlag ( *uint32\_t baseAddr* ) [inline], [static]

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | port base address |
|-----------------|-------------------|

Returns

all 32 pin interrupt status flags. For specific bit:  
• 0: interrupt is not detected.  
• 1: interrupt is detected.

### 38.2.3.9 static void PORT\_HAL\_ClearPortIntFlag ( *uint32\_t baseAddr* ) [inline], [static]

## POR T HAL driver

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>baseAddr</i> | port base address |
|-----------------|-------------------|

# **Chapter 39**

## **Reset Control Module (RCM)**

### **39.1 Overview**

The Kinetis SDK provides a HAL driver for the reset control module (RCM) block of Kinetis devices.

### **Modules**

- RCM HAL driver

## RCM HAL driver

### 39.2 RCM HAL driver

#### 39.2.1 Overview

The RCM implements many of the reset functions for the MCU. See the device reset chapter for more information.

#### 39.2.2 Example

This is an example of the RCM HAL access API functions.

```
#include "rcm/hal/fsl_rcm_hal.h"

printf("\r\nSet the filter control reg for stop mode to LPO\r\n\r\n");
RCM_HAL_SetFilterStopModeCmd(true);

printf("\r\nSet the filter control reg for run/wait mode to LPO\r\n\r\n");
RCM_HAL_SetFilterRunWaitMode(kRcmFilterLpoClk);
```

## Files

- file `fsl_rcm_hal.h`

## Enumerations

- enum `rcm_source_names_t`  
*System Reset Source Name definitions.*
- enum `rcm_filter_run_wait_modes_t`  
*Reset pin filter select in Run and Wait modes.*

## Reset Control Module APIs

- bool `RCM_HAL_GetSrcStatusCmd` (uint32\_t baseAddr, `rcm_source_names_t` srcName)  
*Gets the reset source status.*
- static void `RCM_HAL_SetFilterStopModeCmd` (uint32\_t baseAddr, bool enable)  
*Sets the reset pin filter in stop mode.*
- static bool `RCM_HAL_GetFilterStopModeCmd` (uint32\_t baseAddr)  
*Gets the reset pin filter in stop mode.*
- static void `RCM_HAL_SetFilterRunWaitMode` (uint32\_t baseAddr, `rcm_filter_run_wait_modes_t` mode)  
*Sets the reset pin filter in run and wait mode.*
- static `rcm_filter_run_wait_modes_t` `RCM_HAL_GetFilterRunWaitMode` (uint32\_t baseAddr)  
*Gets the reset pin filter for stop mode.*
- static void `RCM_HAL_SetFilterWidth` (uint32\_t baseAddr, uint32\_t width)  
*Sets the reset pin filter width.*
- static uint32\_t `RCM_HAL_GetFilterWidth` (uint32\_t baseAddr)  
*Gets the reset pin filter for stop mode.*

### 39.2.3 Function Documentation

#### 39.2.3.1 **bool RCM\_HAL\_GetSrcStatusCmd ( uint32\_t *baseAddr*, rcm\_source\_names\_t *srcName* )**

This function gets the current reset source status for a specified source.

## RCM HAL driver

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>baseAddr</i> | Register base address of RCM |
| <i>srcName</i>  | reset source name            |

Returns

status true or false for specified reset source

### **39.2.3.2 static void RCM\_HAL\_SetFilterStopModeCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

This function sets the reset pin filter enable setting in stop mode.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>baseAddr</i> | Register base address of RCM              |
| <i>enable</i>   | enable or disable the filter in stop mode |

### **39.2.3.3 static bool RCM\_HAL\_GetFilterStopModeCmd ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the reset pin filter enable setting in stop mode.

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>baseAddr</i> | Register base address of RCM |
|-----------------|------------------------------|

Returns

enable true/false to enable or disable the filter in stop mode

### **39.2.3.4 static void RCM\_HAL\_SetFilterRunWaitMode ( uint32\_t *baseAddr*, rcm\_filter\_run\_wait\_modes\_t *mode* ) [inline], [static]**

This function sets the reset pin filter enable setting in run/wait mode.

Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>baseAddr</i> | Register base address of RCM                |
| <i>mode</i>     | to be set for reset filter in run/wait mode |

### 39.2.3.5 static rcm\_filter\_run\_wait\_modes\_t RCM\_HAL\_GetFilterRunWaitMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the reset pin filter enable setting for stop mode.

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>baseAddr</i> | Register base address of RCM |
|-----------------|------------------------------|

Returns

mode for reset filter in run/wait mode

### 39.2.3.6 static void RCM\_HAL\_SetFilterWidth ( uint32\_t *baseAddr*, uint32\_t *width* ) [inline], [static]

This function sets the reset pin filter width.

Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>baseAddr</i> | Register base address of RCM     |
| <i>width</i>    | to be set for reset filter width |

### 39.2.3.7 static uint32\_t RCM\_HAL\_GetFilterWidth ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the reset pin filter width.

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>baseAddr</i> | Register base address of RCM |
|-----------------|------------------------------|

Returns

width reset filter width



# **Chapter 40**

## **System Integration Module (SIM)**

### **40.1 Overview**

The Kinetis SDK provides a HAL driver for the System Integration Module (SIM) block of Kinetis devices.

### **Modules**

- [SIM HAL driver](#)

## SIM HAL driver

### 40.2 SIM HAL driver

#### 40.2.1 Overview

The section describes the programming interface of the SIM HAL driver. The system integration module (SIM) provides the system control and device configuration registers. The sim\_hal provides a set of API functions used to access the SIM registers including clock gate control and other configuration settings.

#### 40.2.2 Clock Gate Control register access APIs

Clock gate control is based on the module. Each chip has a sub-set of modules that can be gated through gate control registers in SIM. The gate control names are defined in the enumeration sim\_clock\_gate\_name\_t. Pass the enumeration value and the parameter enables/disables the clock for a module accordingly. There is an example for clock gate APIs:

```
#include "fsl_sim_hal.h"

// Calling sim clock gate control API to enable the clock for UART0 //
SIM_HAL_EnableClock(SIM_BASE, kSimClockGateUart0);

// Calling sim clock gate control API to disable the clock for UART0 //
SIM_HAL_DisableClock(SIM_BASE, kSimClockGateUart0);

// Get the clock gate status for UART0 //
SIM_HAL_GetGateCmd(SIM_BASE, kSimClockGateUart0);
```

#### 40.2.3 Clock Source Control access APIs

Clock source control is also based on the module. Only certain modules have the clock source control in SIM. For these modules, SIM HAL driver provides the separate APIs to set or get module source. The module source setting values are defined in the enumeration with the prefix clock\_. For example, USB FS OTG module uses the MCGPLLFLCLK or the external USB\_CLKIN as a clock source. Therefore, the SIM HAL driver provides these for the USB FS OTG module clock source:

```
typedef enum _clock_usbfs_src
{
 kClockUsbfsSrcExt, // External bypass clock (USB_CLKIN) //
 kClockUsbfsSrcPllFllSel, // MCGPLLFLCLK divided by USB divider //
} clock_usbfs_src_t;

// Set USB FS OTG clock source. //
void CLOCK_HAL_SetUsbfsSrc(uint32_t baseAddr, uint8_t instance, clock_usbfs_src_t
 setting);

// Get USB FS OTG clock source. //
clock_usbfs_src_t CLOCK_HAL_GetUsbfsSrc(uint32_t baseAddr, uint8_t instance);
```

For other IP modules, the clock source selection register is in IP module, but the clock distribution is controlled by the system integration. Therefore, the SIM HAL driver provides the information for the IP modules and IP drivers know which clock sources are available and how to set their internal register to

select a different clock source. This information is provided as an enumeration, such as SAI module using SYSCLK, OSC0ERCLK and MCGPLLCLK as a clock source. The SIM HAL driver provides:

```
typedef enum _clock_sai_src
{
 kClockSaiSrcSysClk = 0U, // SYSCLK //
 kClockSaiSrcOsc0erClk = 1U, // OSC0ERCLK //
 kClockSaiSrcPllClk = 3U // MCGPLLCLK //
} clock_sai_src_t;
```

See the appropriate reference manual for details.

#### 40.2.4 Clock Divider access APIs

Certain clocks use dividers configured in SIM module. The SIM HAL driver provides API functions to get/set the divider values. For example:

```
// To set USB FS OTG divider. //
void CLOCK_HAL_SetUsbfsDiv(uint32_t baseAddr,
 uint8_t usbddiv,
 uint8_t usbfrac);

// To get USB FS OTG divider setting. //
void CLOCK_HAL_GetUsbfsDiv(uint32_t baseAddr,
 uint8_t *usbddiv,
 uint8_t *usbfrac);
```

## Modules

- K02F12810 SIM HAL driver
- K22F12810 SIM HAL driver
- K22F25612 SIM HAL driver
- K22F51212 SIM HAL driver
- K24F12 SIM HAL driver
- K24F25612 SIM HAL driver
- K60D10 SIM HAL driver
- K64F12 SIM HAL driver
- KL03Z4 SIM HAL driver
- KL46Z4 SIM HAL driver
- KV10Z7 SIM HAL driver
- KV30F12810 SIM HAL driver
- KV31F12810 SIM HAL driver
- KV31F25612 SIM HAL driver
- KV31F51212 SIM HAL driver

## Files

- file [fsl\\_sim\\_hal.h](#)
- file [fsl\\_sim\\_hal\\_MK63F12.h](#)
- file [fsl\\_sim\\_hal\\_MKL25Z4.h](#)

### Enumerations

- enum `sim_hal_status_t` {  
  kSimHalSuccess,  
  kSimHalFail }  
    *SIM HAL API return status.*
- enum `sim_sdhc_clock_source_t`  
    *SIM SDHC clock source.*
- enum `sim_time_clock_source_t`  
    *SIM TIME clock source.*
- enum `sim_rmii_clock_source_t`  
    *SIM RMII clock source.*
- enum `sim_usb_clock_source_t`  
    *SIM USB clock source.*
- enum `sim_pllfl_clock_sel_t`  
    *SIM PLLFLLSEL clock source select.*
- enum `sim_osc32k_clock_sel_t`  
    *SIM OSC32KSEL clock source select.*
- enum `sim_trace_clock_sel_t`  
    *SIM TRACESEL clock source select.*
- enum `sim_clkout_clock_sel_t`  
    *SIM CLKOUT\_SEL clock source select.*
- enum `sim_rtclkout_clock_sel_t`  
    *SIM RTCCLKOUTSEL clock source select.*
- enum `sim_usbsstby_stop_t`  
    *SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_stop_t`  
    *SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_cmtuartpad_strenght_t`  
    *SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strenght_t`  
    *SIM PTD7 pad drive strength.*
- enum `sim_flexport_security_level_t`  
    *SIM FlexBus security level.*
- enum `sim_pretrgsel_t`  
    *SIM ADCx pre-trigger select.*
- enum `sim_trgsel_t`  
    *SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_t`  
    *SIM receive data source select.*
- enum `sim_uart_txsrc_t`  
    *SIM transmit data source select.*
- enum `sim_ftm_trg_src_t`  
    *SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_t`  
    *SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_t`  
    *SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftm_ftl_sel_t`  
    *SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_t`

- enum `sim_tpm_ch_src_t`  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_clock_source_t`  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_tpm_clock_source_t`  
*SIM TPM clock source.*
- enum `sim_uart0_clock_source_t`  
*SIM UART0 clock source.*
- enum `sim_usb_clock_source_t`  
*SIM USB clock source.*
- enum `sim_pllfl_clock_sel_t`  
*SIM PLLFLLSEL clock source select.*
- enum `sim_osc32k_clock_sel_t`  
*SIM OSC32KSEL clock source select.*
- enum `sim_clkout_clock_sel_t`  
*SIM CLKOUT\_SEL clock source select.*
- enum `sim_rtclkout_clock_sel_t`  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_usbsstby_stop_t`  
*SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_stop_t`  
*SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_cmtuartpad_strength_t`  
*SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strength_t`  
*SIM PTD7 pad drive strength.*
- enum `sim_flexport_security_level_t`  
*SIM FlexBus security level.*
- enum `sim_pretrgsel_t`  
*SIM ADCx pre-trigger select.*
- enum `sim_trgsel_t`  
*SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_t`  
*SIM receive data source select.*
- enum `sim_uart_txsrc_t`  
*SIM transmit data source select.*
- enum `sim_ftm_trg_src_t`  
*SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_t`  
*SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_t`  
*SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftm_ftl_sel_t`  
*SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_t`  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_t`  
*SIM Timer/PWM x channel y input capture source select.*

### Functions

- static void **SIM\_HAL\_EnableClock** (uint32\_t baseAddr, sim\_clock\_gate\_name\_t name)  
*Enable the clock for specific module.*
- static void **SIM\_HAL\_DisableClock** (uint32\_t baseAddr, sim\_clock\_gate\_name\_t name)  
*Disable the clock for specific module.*
- static bool **SIM\_HAL\_GetGateCmd** (uint32\_t baseAddr, sim\_clock\_gate\_name\_t name)  
*Get the the clock gate state for specific module.*
- static void **CLOCK\_HAL\_SetExternalRefClock32kSrc** (uint32\_t baseAddr, clock\_er32k\_src\_t setting)  
*Set the clock selection of ERCLK32K.*
- static clock\_er32k\_src\_t **CLOCK\_HAL\_GetExternalRefClock32kSrc** (uint32\_t baseAddr)  
*Get the clock selection of ERCLK32K.*
- static void **CLOCK\_HAL\_SetOsc32kOutSel** (uint32\_t baseAddr, clock\_osc32kout\_sel\_t setting)  
*Set OSC32KOUT selection.*
- static clock\_osc32kout\_sel\_t **CLOCK\_HAL\_GetOsc32kOutSel** (uint32\_t baseAddr)  
*Get OSC32KOUT selection.*
- static uint32\_t **SIM\_HAL\_GetRamSize** (uint32\_t baseAddr)  
*Gets RAM size.*
- static void **CLOCK\_HAL\_SetPllf1lSel** (uint32\_t baseAddr, clock\_pllfl\_sel\_t setting)  
*Set PLL/FLL clock selection.*
- static clock\_pllfl\_sel\_t **CLOCK\_HAL\_GetPllf1lSel** (uint32\_t baseAddr)  
*Get PLL/FLL clock selection.*
- static void **CLOCK\_HAL\_SetTraceClkSrc** (uint32\_t baseAddr, clock\_trace\_src\_t setting)  
*Set debug trace clock selection.*
- static clock\_trace\_src\_t **CLOCK\_HAL\_GetTraceClkSrc** (uint32\_t baseAddr)  
*Get debug trace clock selection.*
- static void **CLOCK\_HAL\_SetClkOutSel** (uint32\_t baseAddr, clock\_clkout\_src\_t setting)  
*Set CLKOUTSEL selection.*
- static clock\_clkout\_src\_t **CLOCK\_HAL\_GetClkOutSel** (uint32\_t baseAddr)  
*Get CLKOUTSEL selection.*
- static void **CLOCK\_HAL\_SetOutDiv1** (uint32\_t baseAddr, uint8\_t setting)  
*Set OUTDIV1.*
- static uint8\_t **CLOCK\_HAL\_GetOutDiv1** (uint32\_t baseAddr)  
*Get OUTDIV1.*
- static void **CLOCK\_HAL\_SetOutDiv2** (uint32\_t baseAddr, uint8\_t setting)  
*Set OUTDIV2.*
- static uint8\_t **CLOCK\_HAL\_GetOutDiv2** (uint32\_t baseAddr)  
*Get OUTDIV2.*
- static void **CLOCK\_HAL\_SetOutDiv4** (uint32\_t baseAddr, uint8\_t setting)  
*Set OUTDIV4.*
- static uint8\_t **CLOCK\_HAL\_GetOutDiv4** (uint32\_t baseAddr)  
*Get OUTDIV4.*
- void **CLOCK\_HAL\_SetOutDiv** (uint32\_t baseAddr, uint8\_t outdiv1, uint8\_t outdiv2, uint8\_t outdiv3, uint8\_t outdiv4)  
*Sets the clock out dividers setting.*
- void **CLOCK\_HAL\_GetOutDiv** (uint32\_t baseAddr, uint8\_t \*outdiv1, uint8\_t \*outdiv2, uint8\_t \*outdiv3, uint8\_t \*outdiv4)  
*Gets the clock out dividers setting.*
- void **SIM\_HAL\_SetAdcAlternativeTriggerCmd** (uint32\_t baseAddr, uint32\_t instance, bool enable)  
*Sets the ADCx alternate trigger enable setting.*

- bool **SIM\_HAL\_GetAdcAlternativeTriggerCmd** (uint32\_t baseAddr, uint32\_t instance)
 

*Gets the ADCx alternate trigger enable setting.*
- void **SIM\_HAL\_SetAdcPreTriggerMode** (uint32\_t baseAddr, uint32\_t instance, sim\_adc\_pretrg\_sel\_t select)
 

*Sets the ADCx pre-trigger select setting.*
- sim\_adc\_pretrg\_sel\_t **SIM\_HAL\_GetAdcPreTriggerMode** (uint32\_t baseAddr, uint32\_t instance)
 

*Gets the ADCx pre-trigger select setting.*
- void **SIM\_HAL\_SetAdcTriggerMode** (uint32\_t baseAddr, uint32\_t instance, sim\_adc\_trg\_sel\_t select)
 

*Sets the ADCx trigger select setting.*
- sim\_adc\_trg\_sel\_t **SIM\_HAL\_GetAdcTriggerMode** (uint32\_t baseAddr, uint32\_t instance)
 

*Gets the ADCx trigger select setting.*
- void **SIM\_HAL\_SetAdcTriggerModeOneStep** (uint32\_t baseAddr, uint32\_t instance, bool altTrigEn, sim\_adc\_pretrg\_sel\_t preTrigSel, sim\_adc\_trg\_sel\_t trigSel)
 

*Sets the ADCx trigger select setting in one function.*
- void **SIM\_HAL\_SetUartRxSrcMode** (uint32\_t baseAddr, uint32\_t instance, sim\_uart\_rxsrv\_t select)
 

*Sets the UARTx receive data source select setting.*
- sim\_uart\_rxsrv\_t **SIM\_HAL\_GetUartRxSrcMode** (uint32\_t baseAddr, uint32\_t instance)
 

*Gets the UARTx receive data source select setting.*
- void **SIM\_HAL\_SetUartTxSrcMode** (uint32\_t baseAddr, uint32\_t instance, sim\_uart\_txsrc\_t select)
 

*Sets the UARTx transmit data source select setting.*
- sim\_uart\_txsrc\_t **SIM\_HAL\_GetUartTxSrcMode** (uint32\_t baseAddr, uint32\_t instance)
 

*Gets the UARTx transmit data source select setting.*
- void **SIM\_HAL\_SetFtmTriggerSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t trigger, sim\_ftm\_trg\_src\_t select)
 

*Sets the FlexTimer x hardware trigger y source select setting.*
- sim\_ftm\_trg\_src\_t **SIM\_HAL\_GetFtmTriggerSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t trigger)
 

*Gets the FlexTimer x hardware trigger y source select setting.*
- void **SIM\_HAL\_SetFtmExternalClkPinMode** (uint32\_t baseAddr, uint32\_t instance, sim\_ftm\_clk\_sel\_t select)
 

*Sets the FlexTimer x external clock pin select setting.*
- sim\_ftm\_clk\_sel\_t **SIM\_HAL\_GetFtmExternalClkPinMode** (uint32\_t baseAddr, uint32\_t instance)
 

*Gets the FlexTimer x external clock pin select setting.*
- void **SIM\_HAL\_SetFtmChSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t channel, sim\_ftm\_ch\_src\_t select)
 

*Sets the FlexTimer x channel y input capture source select setting.*
- sim\_ftm\_ch\_src\_t **SIM\_HAL\_GetFtmChSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t channel)
 

*Gets the FlexTimer x channel y input capture source select setting.*
- void **SIM\_HAL\_SetFtmFaultSelMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t fault, sim\_ftm\_ftl\_sel\_t select)
 

*Sets the FlexTimer x fault y select setting.*
- sim\_ftm\_ftl\_sel\_t **SIM\_HAL\_GetFtmFaultSelMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t fault)
 

*Gets the FlexTimer x fault y select setting.*
- void **SIM\_HAL\_SetFtmChOutSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t channel, sim\_ftm\_ch\_out\_src\_t select)
 

*Gets the FlexTimer x channel y output source select setting.*

## SIM HAL driver

- *Sets the FlexTimer x channel y output source select setting.*  
sim\_ftm\_ch\_out\_src\_t **SIM\_HAL\_GetFtmChOutSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t channel)
- *Gets the FlexTimer x channel y output source select setting.*  
void **SIM\_HAL\_SetFtmSyncCmd** (uint32\_t baseAddr, uint32\_t instance, bool sync)
- *Set FlexTimer x hardware trigger 0 software synchronization.*  
static bool **SIM\_HAL\_GetFtmSyncCmd** (uint32\_t baseAddr, uint32\_t instance)
- *Get FlexTimer x hardware trigger 0 software synchronization setting.*  
static uint32\_t **SIM\_HAL\_GetFamilyId** (uint32\_t baseAddr)
- *Gets the Kinetis Family ID in the System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetSubFamilyId** (uint32\_t baseAddr)
- *Gets the Kinetis Sub-Family ID in the System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetSeriesId** (uint32\_t baseAddr)
- *Gets the Kinetis SeriesID in the System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetRevId** (uint32\_t baseAddr)
- *Gets the Kinetis Revision ID in the System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetDieId** (uint32\_t baseAddr)
- *Gets the Kinetis Die ID in the System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetFamId** (uint32\_t baseAddr)
- *Gets the Kinetis family identification in the System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetPinCntId** (uint32\_t baseAddr)
- *Gets the Kinetis Pincount ID in System Device ID register (SIM\_SDID).*  
static uint32\_t **SIM\_HAL\_GetProgramFlashSize** (uint32\_t baseAddr)
- *Gets the program flash size in the Flash Configuration Register 1 (SIM\_FCFG).*  
static void **SIM\_HAL\_SetFlashDoze** (uint32\_t baseAddr, uint32\_t setting)
- *Sets the Flash Doze in the Flash Configuration Register 1 (SIM\_FCFG).*  
static uint32\_t **SIM\_HAL\_GetFlashDoze** (uint32\_t baseAddr)
- *Gets the Flash Doze in the Flash Configuration Register 1 (SIM\_FCFG).*  
static void **SIM\_HAL\_SetFlashDisableCmd** (uint32\_t baseAddr, bool disable)
- *Gets the Flash disable setting.*  
static bool **SIM\_HAL\_GetFlashDisableCmd** (uint32\_t baseAddr)
- *Gets the Flash disable setting.*  
static uint32\_t **SIM\_HAL\_GetFlashMaxAddrBlock0** (uint32\_t baseAddr)
- *Gets the Flash maximum address block 0 in the Flash Configuration Register 1 (SIM\_FCFG).*  
static void **CLOCK\_HAL\_SetUsbfsSrc** (uint32\_t baseAddr, uint32\_t instance, clock\_usbfs\_src\_t setting)
- *Set the selection of the clock source for the USB FS 48 MHz clock.*  
static clock\_usbfs\_src\_t **CLOCK\_HAL\_GetUsbfsSrc** (uint32\_t baseAddr, uint32\_t instance)
- *Get the selection of the clock source for the USB FS 48 MHz clock.*  
void **CLOCK\_HAL\_SetUsbfsDiv** (uint32\_t baseAddr, uint8\_t usbddiv, uint8\_t usbfrac)
- *Set USB FS divider setting.*  
void **CLOCK\_HAL\_GetUsbfsDiv** (uint32\_t baseAddr, uint8\_t \*usbddiv, uint8\_t \*usbfrac)
- *Get USB FS divider setting.*  
static void **CLOCK\_HAL\_SetLpuartSrc** (uint32\_t baseAddr, uint32\_t instance, clock\_lpuart\_src\_t setting)
- *Set LPUART clock source.*  
static clock\_lpuart\_src\_t **CLOCK\_HAL\_GetLpuartSrc** (uint32\_t baseAddr, uint32\_t instance)
- *Get LPUART clock source.*  
static void **CLOCK\_HAL\_SetRtcClkOutSel** (uint32\_t baseAddr, clock\_rtcout\_src\_t setting)
- *Set RTCCLKOUTSEL selection.*  
static clock\_rtcout\_src\_t **CLOCK\_HAL\_GetRtcClkOutSel** (uint32\_t baseAddr)

*Get RTCCLKOUTSEL selection.*

- static void [SIM\\_HAL\\_SetLpuartRxSrcMode](#) (uint32\_t baseAddr, uint32\_t instance, sim\_lpuart\_rxsrv\_t select)

*Sets the LPUARTx receive data source select setting.*

- static sim\_lpuart\_rxsrv\_t [SIM\\_HAL\\_GetLpuartRxSrcMode](#) (uint32\_t baseAddr, uint32\_t instance)

*Gets the LPUARTx receive data source select setting.*

- static uint32\_t [SIM\\_HAL\\_GetFlashMaxAddrBlock1](#) (uint32\_t baseAddr)

*Gets the Flash maximum address block 1 in Flash Configuration Register 2.*

- static void [SIM\\_HAL\\_SetUsbVoltRegulatorCmd](#) (uint32\_t baseAddr, bool enable)

*Sets the USB voltage regulator enabled setting.*

- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorCmd](#) (uint32\_t baseAddr)

*Gets the USB voltage regulator enabled setting.*

- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringStopMode](#) (uint32\_t baseAddr, sim\_usbsstby\_mode\_t setting)

*Sets the USB voltage regulator in a standby mode setting during Stop, VLPS, LLS, and VLLS.*

- static sim\_usbsstby\_mode\_t [SIM\\_HAL\\_GetUsbVoltRegulatorInStdbyDuringStopMode](#) (uint32\_t baseAddr)

*Gets the USB voltage regulator in a standby mode setting.*

- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringVlprwMode](#) (uint32\_t baseAddr, sim\_usbvstby\_mode\_t setting)

*Sets the USB voltage regulator in a standby mode during the VLPR or the VLPW.*

- static sim\_usbvstby\_mode\_t [SIM\\_HAL\\_GetUsbVoltRegulatorInStdbyDuringVlprwMode](#) (uint32\_t baseAddr)

*Gets the USB voltage regulator in a standby mode during the VLPR or the VLPW.*

- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringStopCmd](#) (uint32\_t baseAddr, bool enable)

*Sets the USB voltage regulator stop standby write enable setting.*

- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorInStdbyDuringStopCmd](#) (uint32\_t baseAddr)

*Gets the USB voltage regulator stop standby write enable setting.*

- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringVlprwCmd](#) (uint32\_t baseAddr, bool enable)

*Sets the USB voltage regulator VLP standby write enable setting.*

- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorInStdbyDuringVlprwCmd](#) (uint32\_t baseAddr)

*Gets the USB voltage regulator VLP standby write enable setting.*

- static void [SIM\\_HAL\\_SetUsbVoltRegulatorWriteCmd](#) (uint32\_t baseAddr, bool enable)

*Sets the USB voltage regulator enable write enable setting.*

- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorWriteCmd](#) (uint32\_t baseAddr)

*Gets the USB voltage regulator enable write enable setting.*

- static void [CLOCK\\_HAL\\_SetOutDiv3](#) (uint32\_t baseAddr, uint8\_t setting)

*Set OUTDIV3.*

- static uint8\_t [CLOCK\\_HAL\\_GetOutDiv3](#) (uint32\_t baseAddr)

*Get OUTDIV3.*

- static void [SIM\\_HAL\\_SetFlexbusSecurityLevelMode](#) (uint32\_t baseAddr, sim\_flexbus\_security\_level\_t setting)

*Sets the FlexBus security level setting.*

- static sim\_flexbus\_security\_level\_t [SIM\\_HAL\\_GetFlexbusSecurityLevelMode](#) (uint32\_t baseAddr)

*Gets the FlexBus security level setting.*

- static void [CLOCK\\_HAL\\_SetSdhcSrc](#) (uint32\_t baseAddr, uint32\_t instance, clock\_sdhc\_src\_t setting)

*Set the SDHC clock source selection.*

## SIM HAL driver

- static clock\_sdhc\_src\_t [CLOCK\\_HAL\\_GetSdhcSrc](#) (uint32\_t baseAddr, uint32\_t instance)  
*Get the SDHC clock source selection.*
- static void [SIM\\_HAL\\_SetPtd7PadDriveStrengthMode](#) (uint32\_t baseAddr, [sim\\_ptd7pad\\_strenght\\_t](#) setting)  
*Sets the PTD7 pad drive strength setting.*
- static [sim\\_ptd7pad\\_strenght\\_t](#) [SIM\\_HAL\\_GetPtd7PadDriveStrengthMode](#) (uint32\_t baseAddr)  
*Gets the PTD7 pad drive strength setting.*
- static uint32\_t [SIM\\_HAL\\_GetFlexnvmSize](#) (uint32\_t baseAddr)  
*Gets the FlexNVM size in the Flash Configuration Register 1 (SIM\_FCFG).*
- static uint32\_t [SIM\\_HAL\\_GetEepromSize](#) (uint32\_t baseAddr)  
*Gets the EEPROM size in the Flash Configuration Register 1 (SIM\_FCFG).*
- static uint32\_t [SIM\\_HAL\\_GetFlexnvmPartition](#) (uint32\_t baseAddr)  
*Gets the FlexNVM partition in the Flash Configuration Register 1 (SIM\_FCFG).*
- static uint32\_t [SIM\\_HAL\\_GetProgramFlashCmd](#) (uint32\_t baseAddr)  
*Gets the program flash in the Flash Configuration Register 2.*
- static void [CLOCK\\_HAL\\_SetTimeSrc](#) (uint32\_t baseAddr, uint32\_t instance, [clock\\_time\\_src\\_t](#) setting)  
*Set the ethernet timestamp clock source selection.*
- static clock\_time\_src\_t [CLOCK\\_HAL\\_GetTimeSrc](#) (uint32\_t baseAddr, uint32\_t instance)  
*Get the ethernet timestamp clock source selection.*
- static void [CLOCK\\_HAL\\_SetRmiiSrc](#) (uint32\_t baseAddr, uint32\_t instance, [clock\\_rmii\\_src\\_t](#) setting)  
*Set the Ethernet RMII interface clock source selection.*
- static clock\_rmii\_src\_t [CLOCK\\_HAL\\_GetRmiiSrc](#) (uint32\_t baseAddr, uint32\_t instance)  
*Get the Ethernet RMII interface clock source selection.*
- static bool [SIM\\_HAL\\_GetSwapProgramFlash](#) (uint32\_t baseAddr)  
*Gets the Swap program flash flag in the Flash Configuration Register 2.*
- static void [CLOCK\\_HAL\\_SetTpmSrc](#) (uint32\_t baseAddr, uint32\_t instance, [clock\\_tpm\\_src\\_t](#) setting)  
*Set the clock selection of TPM.*
- static clock\_tpm\_src\_t [CLOCK\\_HAL\\_GetTpmSrc](#) (uint32\_t baseAddr, uint32\_t instance)  
*Get the clock selection of TPM.*
- static void [CLOCK\\_HAL\\_SetCopSrc](#) (uint32\_t baseAddr, [clock\\_cop\\_src\\_t](#) setting)  
*Set the clock selection of COP.*
- static clock\_cop\_src\_t [CLOCK\\_HAL\\_GetCopSrc](#) (uint32\_t baseAddr)  
*Get the clock selection of COP.*
- static void [SIM\\_HAL\\_SetLpUartRxSrcMode](#) (uint32\_t baseAddr, uint32\_t instance, [sim\\_uart\\_rxsrct](#) select)  
*Sets the LPUARTx receive data source select setting.*
- static [sim\\_uart\\_rxsrct](#) [SIM\\_HAL\\_GetLpUartRxSrcMode](#) (uint32\_t baseAddr, uint32\_t instance)  
*Gets the LPUARTx receive data source select setting.*
- static void [SIM\\_HAL\\_SetLpUartTxSrcMode](#) (uint32\_t baseAddr, uint32\_t instance, [sim\\_uart\\_txsrct](#) select)  
*Sets the LPUARTx transmit data source select setting.*
- static [sim\\_uart\\_txsrct](#) [SIM\\_HAL\\_GetLpUartTxSrcMode](#) (uint32\_t baseAddr, uint32\_t instance)  
*Gets the LPUARTx transmit data source select setting.*
- static void [SIM\\_HAL\\_SetLpUartOpenDrainCmd](#) (uint32\_t baseAddr, uint32\_t instance, bool enable)  
*Sets the LPUARTx Open Drain Enable setting.*
- static bool [SIM\\_HAL\\_GetLpUartOpenDrainCmd](#) (uint32\_t baseAddr, uint32\_t instance)

- Gets the LPUARTx Open Drain Enable setting.
- static void **SIM\_HAL\_SetTpmChSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t channel, sim\_tpm\_ch\_src\_t select)
  - Sets the Timer/PWM x channel y input capture source select setting.
- static sim\_tpm\_ch\_src\_t **SIM\_HAL\_GetTpmChSrcMode** (uint32\_t baseAddr, uint32\_t instance, uint8\_t channel)
  - Gets the Timer/PWM x channel y input capture source select setting.
- void **SIM\_HAL\_SetTpmExternalClkPinSelMode** (uint32\_t baseAddr, uint32\_t instance, sim\_tpm\_clk\_sel\_t select)
  - Sets the Timer/PWM x external clock pin select setting.
- sim\_tpm\_clk\_sel\_t **SIM\_HAL\_GetTpmExternalClkPinSelMode** (uint32\_t baseAddr, uint32\_t instance)
  - Gets the Timer/PWM x external clock pin select setting.
- static void **CLOCK\_HAL\_SetLpisciSrc** (uint32\_t baseAddr, uint32\_t instance, clock\_lpisci\_src\_t setting)
  - Set the LPSCI clock source selection.
- static clock\_lpisci\_src\_t **CLOCK\_HAL\_GetLpisciSrc** (uint32\_t baseAddr, uint32\_t instance)
  - Get the LPSCI clock source selection.
- static uint32\_t **SIM\_HAL\_GetSramSize** (uint32\_t baseAddr)
  - Gets the Kinetis SramSize in the System Device ID register (SIM\_SDID).
- static void **CLOCK\_HAL\_SetOutDiv5ENCmd** (uint32\_t baseAddr, bool setting)
  - Set OUTDIV5EN.
- static bool **CLOCK\_HAL\_GetOutDiv5ENCmd** (uint32\_t baseAddr)
  - Get OUTDIV5EN.
- static void **CLOCK\_HAL\_SetOutDiv5** (uint32\_t baseAddr, uint8\_t setting)
  - Set OUTDIV5.
- static uint8\_t **CLOCK\_HAL\_GetOutDiv5** (uint32\_t baseAddr)
  - Get OUTDIV5.
- void **CLOCK\_HAL\_SetAdcAltClkSrc** (uint32\_t baseAddr, uint32\_t instance, clock\_adc\_alt\_src\_t adcAltSrcSel)
  - Sets the ADC ALT clock source selection setting.
- clock\_adc\_alt\_src\_t **CLOCK\_HAL\_GetAdcAltClkSrc** (uint32\_t baseAddr, uint32\_t instance)
  - Gets the ADC ALT clock source selection setting.
- void **SIM\_HAL\_SetUartOpenDrainMode** (uint32\_t baseAddr, uint32\_t instance, bool enable)
  - Sets the UARTx open drain enable setting.
- bool **SIM\_HAL\_GetUartOpenDrainMode** (uint32\_t baseAddr, uint32\_t instance)
  - Gets the UARTx open drain enable setting.
- static void **CLOCK\_HAL\_SetFtmFixFreqClkSrc** (uint32\_t baseAddr, clock\_ftm\_fixedfreq\_src\_t ftmFixedFreqSel)
  - Sets the FTM Fixed clock source selection setting.
- static clock\_ftm\_fixedfreq\_src\_t **CLOCK\_HAL\_GetFtmFixFreqClkSrc** (uint32\_t baseAddr)
  - Gets the FTM Fixed clock source selection setting.
- static void **SIM\_HAL\_SetFtmCarrierFreqMode** (uint32\_t baseAddr, sim\_ftm\_ftl\_carrier\_sel\_t select)
  - Sets the Carrier frequency selection for FTM0/2 output channel.
- static sim\_ftm\_ftl\_carrier\_sel\_t **SIM\_HAL\_GetFtmCarrierFreqMode** (uint32\_t baseAddr)
  - Gets the Carrier frequency selection for FTM0/2 output channel.
- static uint32\_t **SIM\_HAL\_GetSubFamId** (uint32\_t baseAddr)
  - Gets the Kinetis SbuFam ID in System Device ID register (SIM\_SDID).
- static uint32\_t **SIM\_HAL\_GetSramSizeId** (uint32\_t baseAddr)

## SIM HAL driver

- Gets the Kinetis SRAMSIZE ID in the System Device ID register (SIM\_SDID).
- static uint32\_t **SIM\_HAL\_GetFlashMaxAddrBlock** (uint32\_t baseAddr)  
    Gets the Flash maximum address block in the Flash Configuration Register 1 (SIM\_FCFG).

## IP related clock feature APIs

- void **SIM\_HAL\_EnableDmaClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for DMA module.
- void **SIM\_HAL\_DisableDmaClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for DMA module.
- bool **SIM\_HAL\_GetDmaGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for DMA module.
- void **SIM\_HAL\_EnableDmamuxClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for DMAMUX module.
- void **SIM\_HAL\_DisableDmamuxClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for DMAMUX module.
- bool **SIM\_HAL\_GetDmamuxGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for DMAMUX module.
- void **SIM\_HAL\_EnablePortClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for PORT module.
- void **SIM\_HAL\_DisablePortClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for PORT module.
- bool **SIM\_HAL\_GetPortGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for PORT module.
- void **SIM\_HAL\_EnableMpuClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for MPU module.
- void **SIM\_HAL\_DisableMpuClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for MPU module.
- bool **SIM\_HAL\_GetMpuGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for MPU module.
- void **SIM\_HAL\_EnableEwmClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for EWM module.
- void **SIM\_HAL\_DisableEwmClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for EWM module.
- bool **SIM\_HAL\_GetEwmGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for EWM module.
- void **SIM\_HAL\_EnableFlexbusClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for FLEXBUS module.
- void **SIM\_HAL\_DisableFlexbusClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for FLEXBUS module.
- bool **SIM\_HAL\_GetFlexbusGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for FLEXBUS module.
- void **SIM\_HAL\_EnableFtfClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for FTF module.
- void **SIM\_HAL\_DisableFtfClock** (uint32\_t baseAddr, uint32\_t instance)  
    Disable the clock for FTF module.
- bool **SIM\_HAL\_GetFtfGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
    Get the the clock gate state for FTF module.
- void **SIM\_HAL\_EnableCrcClock** (uint32\_t baseAddr, uint32\_t instance)  
    Enable the clock for CRC module.

- void **SIM\_HAL\_DisableCrcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for CRC module.*
- bool **SIM\_HAL\_GetCrcGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for CRC module.*
- void **SIM\_HAL\_EnableRngaClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for RNGA module.*
- void **SIM\_HAL\_DisableRngaClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for RNGA module.*
- bool **SIM\_HAL\_GetRngaGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for RNGA module.*
- void **SIM\_HAL\_EnableAdcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for ADC module.*
- void **SIM\_HAL\_DisableAdcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for ADC module.*
- bool **SIM\_HAL\_GetAdcGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for ADC module.*
- void **SIM\_HAL\_EnableCmpClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for CMP module.*
- void **SIM\_HAL\_DisableCmpClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for CMP module.*
- bool **SIM\_HAL\_GetCmpGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for CMP module.*
- void **SIM\_HAL\_EnableDacClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for DAC module.*
- void **SIM\_HAL\_DisableDacClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for DAC module.*
- bool **SIM\_HAL\_GetDacGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for DAC module.*
- void **SIM\_HAL\_EnableVrefClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for VREF module.*
- void **SIM\_HAL\_DisableVrefClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for VREF module.*
- bool **SIM\_HAL\_GetVrefGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for VREF module.*
- void **SIM\_HAL\_EnableSaiClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for SAI module.*
- void **SIM\_HAL\_DisableSaiClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for SAI module.*
- bool **SIM\_HAL\_GetSaiGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for SAI module.*
- void **SIM\_HAL\_EnablePdbClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for PDB module.*
- void **SIM\_HAL\_DisablePdbClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for PDB module.*
- bool **SIM\_HAL\_GetPdbGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for PDB module.*
- void **SIM\_HAL\_EnableFtmClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for FTM module.*
- void **SIM\_HAL\_DisableFtmClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for FTM module.*
- bool **SIM\_HAL\_GetFtmGateCmd** (uint32\_t baseAddr, uint32\_t instance)

## SIM HAL driver

*Get the the clock gate state for FTM module.*

- void **SIM\_HAL\_EnablePitClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for PIT module.*
- void **SIM\_HAL\_DisablePitClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for PIT module.*
- bool **SIM\_HAL\_GetPitGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for PIT module.*
- void **SIM\_HAL\_EnableLptimerClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for LPTIMER module.*
- void **SIM\_HAL\_DisableLptimerClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for LPTIMER module.*
- bool **SIM\_HAL\_GetLptimerGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for LPTIMER module.*
- void **SIM\_HAL\_EnableCmtClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for CMT module.*
- void **SIM\_HAL\_DisableCmtClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for CMT module.*
- bool **SIM\_HAL\_GetCmtGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for CMT module.*
- void **SIM\_HAL\_EnableRtcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for RTC module.*
- void **SIM\_HAL\_DisableRtcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for RTC module.*
- bool **SIM\_HAL\_GetRtcGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for RTC module.*
- void **SIM\_HAL\_EnableEnetClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for ENET module.*
- void **SIM\_HAL\_DisableEnetClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for ENET module.*
- bool **SIM\_HAL\_GetEnetGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for ENET module.*
- void **SIM\_HAL\_EnableUsbClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for USBFS module.*
- void **SIM\_HAL\_DisableUsbClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for USBFS module.*
- bool **SIM\_HAL\_GetUsbGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for USB module.*
- void **SIM\_HAL\_EnableUsbcdClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for USBD\_CD module.*
- void **SIM\_HAL\_DisableUsbcdClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for USBD\_CD module.*
- bool **SIM\_HAL\_GetUsbcdGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for USBD\_CD module.*
- void **SIM\_HAL\_EnableFlexcanClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for FLEXCAN module.*
- void **SIM\_HAL\_DisableFlexcanClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for FLEXCAN module.*
- bool **SIM\_HAL\_GetFlexcanGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for FLEXCAN module.*
- void **SIM\_HAL\_EnableSpiClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for SPI module.*

- void **SIM\_HAL\_DisableSpiClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for SPI module.*
- bool **SIM\_HAL\_GetSpiGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for SPI module.*
- void **SIM\_HAL\_EnableI2cClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for I2C module.*
- void **SIM\_HAL\_DisableI2cClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for I2C module.*
- bool **SIM\_HAL\_GetI2cGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for I2C module.*
- void **SIM\_HAL\_EnableUartClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for UART module.*
- void **SIM\_HAL\_DisableUartClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for UART module.*
- bool **SIM\_HAL\_GetUartGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for UART module.*
- void **SIM\_HAL\_EnableSdhcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for SDHC module.*
- void **SIM\_HAL\_DisableSdhcClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for SDHC module.*
- bool **SIM\_HAL\_GetSdhcGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for SDHC module.*

## IP related clock feature APIs

- void **SIM\_HAL\_EnableTpmClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for TPM module.*
- void **SIM\_HAL\_DisableTpmClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for TPM module.*
- bool **SIM\_HAL\_GetTpmGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for TPM module.*
- void **SIM\_HAL\_EnableTsiClock** (uint32\_t baseAddr, uint32\_t instance)  
*Enable the clock for TSI module.*
- void **SIM\_HAL\_DisableTsiClock** (uint32\_t baseAddr, uint32\_t instance)  
*Disable the clock for TSI module.*
- bool **SIM\_HAL\_GetTsiGateCmd** (uint32\_t baseAddr, uint32\_t instance)  
*Get the the clock gate state for TSI module.*

### 40.2.5 Enumeration Type Documentation

#### 40.2.5.1 enum sim\_hal\_status\_t

Enumerator

**kSimHalSuccess** Success.

**kSimHalFail** Error occurs.

## 40.2.6 Function Documentation

**40.2.6.1 static void SIM\_HAL\_EnableClock ( uint32\_t *baseAddr*, sim\_clock\_gate\_name\_t *name* ) [inline], [static]**

This function enables the clock for specific module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>name</i>     | Name of the module to enable.          |

#### 40.2.6.2 static void SIM\_HAL\_DisableClock ( *uint32\_t baseAddr, sim\_clock\_gate\_name\_t name* ) [inline], [static]

This function disables the clock for specific module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>name</i>     | Name of the module to disable.         |

#### 40.2.6.3 static bool SIM\_HAL\_GetGateCmd ( *uint32\_t baseAddr, sim\_clock\_gate\_name\_t name* ) [inline], [static]

This function will get the clock gate state for specific module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>name</i>     | Name of the module to get.             |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.4 static void CLOCK\_HAL\_SetExternalRefClock32kSrc ( *uint32\_t baseAddr, clock\_er32k\_src\_t setting* ) [inline], [static]

This function sets the clock selection of ERCLK32K.

Parameters

## SIM HAL driver

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

**40.2.6.5 static clock\_er32k\_src\_t CLOCK\_HAL\_GetExternalRefClock32kSrc ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the clock selection of ERCLK32K.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

**40.2.6.6 static void CLOCK\_HAL\_SetOsc32kOutSel ( uint32\_t *baseAddr*, clock\_osc32kout\_sel\_t *setting* ) [inline], [static]**

This function sets ERCLK32K output pin.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

**40.2.6.7 static clock\_osc32kout\_sel\_t CLOCK\_HAL\_GetOsc32kOutSel ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets ERCLK32K output pin setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

**40.2.6.8 static uint32\_t SIM\_HAL\_GetRamSize ( uint32\_t *baseAddr* ) [inline],  
[static]**

This function gets the RAM size. The field specifies the amount of system RAM available on the device.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

size RAM size on the device

### 40.2.6.9 static void CLOCK\_HAL\_SetPllfllSel ( *uint32\_t baseAddr, clock\_pllfill\_sel\_t setting* ) [inline], [static]

This function sets the selection of the high frequency clock for various peripheral clocking options

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

### 40.2.6.10 static *clock\_pllfill\_sel\_t* CLOCK\_HAL\_GetPllfllSel ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the selection of the high frequency clock for various peripheral clocking options

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

### 40.2.6.11 static void CLOCK\_HAL\_SetTraceClkSrc ( *uint32\_t baseAddr, clock\_trace\_src\_t setting* ) [inline], [static]

This function sets debug trace clock selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

**40.2.6.12 static clock\_trace\_src\_t CLOCK\_HAL\_GetTraceClkSrc ( uint32\_t *baseAddr* )  
[inline], [static]**

This function gets debug trace clock selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

**40.2.6.13 static void CLOCK\_HAL\_SetClkOutSel ( uint32\_t *baseAddr*, clock\_clkout\_src\_t  
setting ) [inline], [static]**

This function sets the selection of the clock to output on the CLKOUT pin.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

**40.2.6.14 static clock\_clkout\_src\_t CLOCK\_HAL\_GetClkOutSel ( uint32\_t *baseAddr* )  
[inline], [static]**

This function gets the selection of the clock to output on the CLKOUT pin.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

## SIM HAL driver

**40.2.6.15 static void CLOCK\_HAL\_SetOutDiv1 ( uint32\_t *baseAddr*, uint8\_t *setting* )  
[inline], [static]**

This function sets divide value OUTDIV1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.16 static uint8\_t CLOCK\_HAL\_GetOutDiv1 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets divide value OUTDIV1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current divide value.

#### 40.2.6.17 static void CLOCK\_HAL\_SetOutDiv2 ( uint32\_t *baseAddr*, uint8\_t *setting* ) [inline], [static]

This function sets divide value OUTDIV2.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.18 static uint8\_t CLOCK\_HAL\_GetOutDiv2 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets divide value OUTDIV2.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current divide value.

## SIM HAL driver

**40.2.6.19 static void CLOCK\_HAL\_SetOutDiv4 ( uint32\_t *baseAddr*, uint8\_t *setting* )  
[inline], [static]**

This function sets divide value OUTDIV4.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.20 static uint8\_t CLOCK\_HAL\_GetOutDiv4 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets divide value OUTDIV4.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current divide value.

#### 40.2.6.21 void CLOCK\_HAL\_SetOutDiv ( uint32\_t *baseAddr*, uint8\_t *outdiv1*, uint8\_t *outdiv2*, uint8\_t *outdiv3*, uint8\_t *outdiv4* )

This function sets the setting for all clock out dividers at the same time.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>outdiv1</i>  | Outdivider1 setting                    |
| <i>outdiv2</i>  | Outdivider2 setting                    |
| <i>outdiv3</i>  | Outdivider3 setting                    |
| <i>outdiv4</i>  | Outdivider4 setting                    |

#### 40.2.6.22 void CLOCK\_HAL\_GetOutDiv ( uint32\_t *baseAddr*, uint8\_t \* *outdiv1*, uint8\_t \* *outdiv2*, uint8\_t \* *outdiv3*, uint8\_t \* *outdiv4* )

This function gets the setting for all clock out dividers at the same time.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>outdiv1</i>  | Outdivider1 setting                    |
| <i>outdiv2</i>  | Outdivider2 setting                    |
| <i>outdiv3</i>  | Outdivider3 setting                    |
| <i>outdiv4</i>  | Outdivider4 setting                    |

### 40.2.6.23 void SIM\_HAL\_SetAdcAlternativeTriggerCmd ( uint32\_t *baseAddr*, uint32\_t *instance*, bool *enable* ) [inline]

This function enables/disables the alternative conversion triggers for ADCx.

Parameters

|                 |                                                                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                              |
| <i>instance</i> | device instance.                                                                                                                                                                    |
| <i>enable</i>   | Enable alternative conversion triggers for ADCx <ul style="list-style-type: none"><li>• true: Select alternative conversion trigger.</li><li>• false: Select PDB trigger.</li></ul> |

### 40.2.6.24 bool SIM\_HAL\_GetAdcAlternativeTriggerCmd ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline]

This function gets the ADCx alternate trigger enable setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

enabled True if ADCx alternate trigger is enabled

#### 40.2.6.25 void SIM\_HAL\_SetAdcPreTriggerMode ( *uint32\_t baseAddr, uint32\_t instance, sim\_adc\_pretrg\_sel\_t select* ) [inline]

This function selects the ADCx pre-trigger source when the alternative triggers are enabled through ADCxALTTRGEN.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | pre-trigger select setting for ADCx    |

**40.2.6.26 sim\_adc\_pretrg\_sel\_t SIM\_HAL\_GetAdcPreTriggerMode ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline]**

This function gets the ADCx pre-trigger select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select ADCx pre-trigger select setting

**40.2.6.27 void SIM\_HAL\_SetAdcTriggerMode ( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_adc\_trg\_sel\_t *select* ) [inline]**

This function selects the ADCx trigger source when alternative triggers are enabled through ADCxALT-TRGEN.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | trigger select setting for ADCx        |

**40.2.6.28 sim\_adc\_trg\_sel\_t SIM\_HAL\_GetAdcTriggerMode ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline]**

This function gets the ADCx trigger select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

ADCx trigger select setting

**40.2.6.29 void SIM\_HAL\_SetAdcTriggerModeOneStep ( uint32\_t *baseAddr*, uint32\_t *instance*, bool *altTrigEn*, sim\_adc\_pretrg\_sel\_t *preTrigSel*, sim\_adc\_trg\_sel\_t *trigSel* )**

This function sets ADC alternate trigger, pre-trigger mode and trigger mode.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <i>baseAddr</i>   | Base address for current SIM instance. |
| <i>instance</i>   | device instance.                       |
| <i>altTrigEn</i>  | Alternative trigger enable or not.     |
| <i>preTrigSel</i> | Pre-trigger mode.                      |
| <i>trigSel</i>    | Trigger mode.                          |

**40.2.6.30 void SIM\_HAL\_SetUartRxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_uart\_rxsrc\_t *select* )**

This function selects the source for the UARTx receive data.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | the source for the UARTx receive data  |

**40.2.6.31 sim\_uart\_rxsrc\_t SIM\_HAL\_GetUartRxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function gets the UARTx receive data source select setting.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select UARTx receive data source select setting

### 40.2.6.32 void SIM\_HAL\_SetUartTxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_uart\_txsrc\_t *select* )

This function selects the source for the UARTx transmit data.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | the source for the UARTx transmit data |

### 40.2.6.33 sim\_uart\_txsrc\_t SIM\_HAL\_GetUartTxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function gets the UARTx transmit data source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select UARTx transmit data source select setting

### 40.2.6.34 void SIM\_HAL\_SetFtmTriggerSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, uint8\_t *trigger*, sim\_ftm\_trg\_src\_t *select* )

This function selects the source of FTMx hardware trigger y.

Parameters

|                 |                                                                                                                                                                          |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                   |
| <i>instance</i> | device instance.                                                                                                                                                         |
| <i>trigger</i>  | hardware trigger y                                                                                                                                                       |
| <i>select</i>   | FlexTimer x hardware trigger y<br><ul style="list-style-type: none"> <li>• 0: Pre-trigger A selected for ADCx.</li> <li>• 1: Pre-trigger B selected for ADCx.</li> </ul> |

#### 40.2.6.35 `sim_ftm_trg_src_t SIM_HAL_GetFtmTriggerSrcMode ( uint32_t baseAddr, uint32_t instance, uint8_t trigger )`

This function gets the FlexTimer x hardware trigger y source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>trigger</i>  | hardware trigger y                     |

Returns

select FlexTimer x hardware trigger y source select setting

#### 40.2.6.36 `void SIM_HAL_SetFtmExternalClkPinMode ( uint32_t baseAddr, uint32_t instance, sim_ftm_clk_sel_t select )`

This function selects the source of FTMx external clock pin select.

Parameters

|                 |                                                                                                                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                             |
| <i>instance</i> | device instance.                                                                                                                                                                                   |
| <i>select</i>   | FTMx external clock pin select<br><ul style="list-style-type: none"> <li>• 0: FTMx external clock driven by FTM CLKIN0 pin.</li> <li>• 1: FTMx external clock driven by FTM CLKIN1 pin.</li> </ul> |

## SIM HAL driver

**40.2.6.37 sim\_ftm\_clk\_sel\_t SIM\_HAL\_GetFtmExternalClkPinMode ( uint32\_t *baseAddr*,  
                  uint32\_t *instance* )**

This function gets the FlexTimer x external clock pin select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select FlexTimer x external clock pin select setting

#### 40.2.6.38 void SIM\_HAL\_SetFtmChSrcMode ( *uint32\_t baseAddr, uint32\_t instance, uint8\_t channel, sim\_ftm\_ch\_src\_t select* )

This function selects the FlexTimer x channel y input capture source.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.     |
| <i>instance</i> | device instance.                           |
| <i>channel</i>  | FlexTimer channel y                        |
| <i>select</i>   | FlexTimer x channel y input capture source |

#### 40.2.6.39 sim\_ftm\_ch\_src\_t SIM\_HAL\_GetFtmChSrcMode ( *uint32\_t baseAddr, uint32\_t instance, uint8\_t channel* )

This function gets the FlexTimer x channel y input capture source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>channel</i>  | FlexTimer channel y                    |

Returns

select FlexTimer x channel y input capture source select setting

#### 40.2.6.40 void SIM\_HAL\_SetFtmFaultSelMode ( *uint32\_t baseAddr, uint32\_t instance, uint8\_t fault, sim\_ftm\_ftl\_sel\_t select* )

This function sets the FlexTimer x fault y select setting.

## SIM HAL driver

Parameters

|                 |                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                           |
| <i>instance</i> | device instance.                                                                                                                                                 |
| <i>fault</i>    | fault y                                                                                                                                                          |
| <i>select</i>   | FlexTimer x fault y select setting <ul style="list-style-type: none"><li>• 0: FlexTimer x fault y select 0.</li><li>• 1: FlexTimer x fault y select 1.</li></ul> |

### 40.2.6.41 sim\_ftm\_ftl\_sel\_t SIM\_HAL\_GetFtmFaultSelMode ( uint32\_t *baseAddr*, uint32\_t *instance*, uint8\_t *fault* )

This function gets the FlexTimer x fault y select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>fault</i>    | fault y                                |

Returns

select FlexTimer x fault y select setting

### 40.2.6.42 void SIM\_HAL\_SetFtmChOutSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, uint8\_t *channel*, sim\_ftm\_ch\_out\_src\_t *select* )

This function selects the FlexTimer x channel y output source.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>channel</i>  | FlexTimer channel y                    |

|               |                                     |
|---------------|-------------------------------------|
| <i>select</i> | FlexTimer x channel y output source |
|---------------|-------------------------------------|

#### 40.2.6.43 **sim\_ftm\_ch\_out\_src\_t SIM\_HAL\_GetFtmChOutSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, uint8\_t *channel* )**

This function gets the FlexTimer x channel y output source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>channel</i>  | FlexTimer channel y                    |

Returns

select FlexTimer x channel y output source select setting

#### 40.2.6.44 **void SIM\_HAL\_SetFtmSyncCmd ( uint32\_t *baseAddr*, uint32\_t *instance*, bool *sync* )**

This function sets FlexTimer x hardware trigger 0 software synchronization. FTMXSYNCBIT.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>sync</i>     | Synchronize or not.                    |

#### 40.2.6.45 **static bool SIM\_HAL\_GetFtmSyncCmd ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets FlexTimer x hardware trigger 0 software synchronization. FTMXSYNCBIT.

Parameters

## SIM HAL driver

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

**40.2.6.46 static uint32\_t SIM\_HAL\_GetFamilyId ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Kinetis Family ID in the System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis Family ID

**40.2.6.47 static uint32\_t SIM\_HAL\_GetSubFamilyId ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Kinetis Sub-Family ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis Sub-Family ID

**40.2.6.48 static uint32\_t SIM\_HAL\_GetSeriesId ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Kinetis Series ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis Series ID

#### 40.2.6.49 static uint32\_t SIM\_HAL\_GetRevId ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Kinetis Revision ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis Revision ID

#### 40.2.6.50 static uint32\_t SIM\_HAL\_GetDieId ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Kinetis Die ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis Die ID

#### 40.2.6.51 static uint32\_t SIM\_HAL\_GetFamId ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Kinetis family identification in System Device ID register.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

*id* Kinetis family identification

### 40.2.6.52 static uint32\_t SIM\_HAL\_GetPinCntId ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Kinetis Pincount ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

*id* Kinetis Pincount ID

### 40.2.6.53 static uint32\_t SIM\_HAL\_GetProgramFlashSize ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the program flash size in the Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

*size* Program flash Size

### 40.2.6.54 static void SIM\_HAL\_SetFlashDoze ( uint32\_t *baseAddr*, uint32\_t *setting* ) [inline], [static]

This function sets the Flash Doze in the Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | Flash Doze setting                     |

#### 40.2.6.55 static uint32\_t SIM\_HAL\_GetFlashDoze ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Flash Doze in the Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting Flash Doze setting

#### 40.2.6.56 static void SIM\_HAL\_SetFlashDisableCmd ( uint32\_t *baseAddr*, bool *disable* ) [inline], [static]

This function sets the Flash disable setting in the Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>disable</i>  | Flash disable setting                  |

#### 40.2.6.57 static bool SIM\_HAL\_GetFlashDisableCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Flash disable setting in the Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting Flash disable setting

## SIM HAL driver

**40.2.6.58 static uint32\_t SIM\_HAL\_GetFlashMaxAddrBlock0 ( uint32\_t *baseAddr* )  
[inline], [static]**

This function gets the Flash maximum block 0 in Flash Configuration Register 2.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

address Flash maximum block 0 address

#### 40.2.6.59 static void CLOCK\_HAL\_SetUsbfsSrc ( *uint32\_t baseAddr, uint32\_t instance, clock\_usbfs\_src\_t setting* ) [inline], [static]

This function sets the selection of the clock source for the USB FS 48 MHz clock.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.60 static *clock\_usbfs\_src\_t* CLOCK\_HAL\_GetUsbfsSrc ( *uint32\_t baseAddr, uint32\_t instance* ) [inline], [static]

This function gets the selection of the clock source for the USB FS 48 MHz clock.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

#### 40.2.6.61 void CLOCK\_HAL\_SetUsbfsDiv ( *uint32\_t baseAddr, uint8\_t usbddiv, uint8\_t usbfrac* )

This function sets USB FS divider setting. Divider output clock = Divider input clock \* [ (USBFSFRA-C+1) / (USBFSDIV+1) ]

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>usbdiv</i>   | Value of USBFSDIV.                     |
| <i>usbfrac</i>  | Value of USBFSFRAC.                    |

**40.2.6.62 void CLOCK\_HAL\_GetUsbfsDiv ( uint32\_t *baseAddr*, uint8\_t \* *usbdiv*, uint8\_t \* *usbfrac* )**

This function gets USB FS divider setting. Divider output clock = Divider input clock \* [ (USBFSFRA-C+1) / (USBFSDIV+1) ]

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>usbdiv</i>   | Value of USBFSDIV.                     |
| <i>usbfrac</i>  | Value of USBFSFRAC.                    |

**40.2.6.63 static void CLOCK\_HAL\_SetLpuartSrc ( uint32\_t *baseAddr*, uint32\_t *instance*, clock\_lpuart\_src\_t *setting* ) [inline], [static]**

This function sets lpuart clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | LPUART instance.                       |
| <i>setting</i>  | The value to set.                      |

**40.2.6.64 static clock\_lpuart\_src\_t CLOCK\_HAL\_GetLpuartSrc ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets lpuart clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | LPUART instance.                       |

Returns

Current selection.

#### 40.2.6.65 static void CLOCK\_HAL\_SetRtcClkOutSel ( uint32\_t *baseAddr*, clock\_rtcout\_src\_t *setting* ) [inline], [static]

This function sets the selection of the clock to output on the RTC\_CLKOUT pin.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.66 static clock\_rtcout\_src\_t CLOCK\_HAL\_GetRtcClkOutSel ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the selection of the clock to output on the RTC\_CLKOUT pin.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

#### 40.2.6.67 static void SIM\_HAL\_SetLpuartRxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_lpuart\_rxsrc\_t *select* ) [inline], [static]

This function selects the source for the LPUARTx receive data.

Parameters

## SIM HAL driver

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.  |
| <i>instance</i> | device instance.                        |
| <i>select</i>   | the source for the LPUARTx receive data |

**40.2.6.68 static sim\_lpuart\_rxsrc\_t SIM\_HAL\_GetLpuartRxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets the LPUARTx receive data source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select LPUARTx receive data source select setting

**40.2.6.69 static uint32\_t SIM\_HAL\_GetFlashMaxAddrBlock1 ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Flash maximum block 1 in Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

address Flash maximum block 0 address

**40.2.6.70 static void SIM\_HAL\_SetUsbVoltRegulatorCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]**

This function controls whether the USB voltage regulator is enabled. This bit can only be written when the SOPT1CFG[URWE] bit is set.

Parameters

|                 |                                                                                                                                                                                       |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                |
| <i>enable</i>   | USB voltage regulator enable setting <ul style="list-style-type: none"> <li>• true: USB voltage regulator is enabled.</li> <li>• false: USB voltage regulator is disabled.</li> </ul> |

#### 40.2.6.71 static bool SIM\_HAL\_GetUsbVoltRegulatorCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the USB voltage regulator enabled setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

*enabled* True if the USB voltage regulator is enabled.

#### 40.2.6.72 static void SIM\_HAL\_SetUsbVoltRegulatorInStandbyDuringStopMode ( uint32\_t *baseAddr*, sim\_usbsstby\_mode\_t *setting* ) [inline], [static]

This function controls whether the USB voltage regulator is placed in a standby mode during Stop, VLPS, LLS, and VLLS modes. This bit can only be written when the SOPT1CFG[USSWE] bit is set.

Parameters

|                 |                                                                                                                                                                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                                                                                 |
| <i>setting</i>  | USB voltage regulator in standby mode setting <ul style="list-style-type: none"> <li>• 0: USB voltage regulator not in standby during Stop, VLPS, LLS and VLLS modes.</li> <li>• 1: USB voltage regulator in standby during Stop, VLPS, LLS and VLLS modes.</li> </ul> |

#### 40.2.6.73 static sim\_usbsstby\_mode\_t SIM\_HAL\_GetUsbVoltRegulator- InStandbyDuringStopMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the USB voltage regulator in a standby mode setting.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting USB voltage regulator in a standby mode setting

### 40.2.6.74 static void SIM\_HAL\_SetUsbVoltRegulatorInStdbyDuringVlprwMode ( uint32\_t *baseAddr*, sim\_usbvstby\_mode\_t *setting* ) [inline], [static]

This function controls whether the USB voltage regulator is placed in a standby mode during the VLPR and the VLPW modes. This bit can only be written when the SOPT1CFG[UVSWE] bit is set.

Parameters

|                 |                                                                                                                                                                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                                                        |
| <i>setting</i>  | USB voltage regulator in standby mode setting <ul style="list-style-type: none"><li>• 0: USB voltage regulator not in standby during VLPR and VLPW modes.</li><li>• 1: USB voltage regulator in standby during VLPR and VLPW modes.</li></ul> |

### 40.2.6.75 static sim\_usbvstby\_mode\_t SIM\_HAL\_GetUsbVoltRegulator- InStdbyDuringVlprwMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the USB voltage regulator in a standby mode during the VLPR or the VLPW.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting USB voltage regulator in a standby mode during the VLPR or the VLPW

### 40.2.6.76 static void SIM\_HAL\_SetUsbVoltRegulatorInStdbyDuringStopCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function controls whether the USB voltage regulator stop standby write feature is enabled. Writing one to this bit allows the SOPT1[USBSSTBY] bit to be written. This register bit clears after a write to SOPT1[USBSSTBY].

Parameters

|                 |                                                                                                                                                                                                        |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                 |
| <i>enable</i>   | USB voltage regulator stop standby write enable setting <ul style="list-style-type: none"> <li>• true: SOPT1[USBSSTBY] can be written.</li> <li>• false: SOPT1[USBSSTBY] cannot be written.</li> </ul> |

#### 40.2.6.77 static bool SIM\_HAL\_GetUsbVoltRegulatorInStdbyDuringStopCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the USB voltage regulator stop standby write enable setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

*enabled* True if the USB voltage regulator stop standby write is enabled.

#### 40.2.6.78 static void SIM\_HAL\_SetUsbVoltRegulatorInStdbyDuringVlprwCmd ( uint32\_t *baseAddr*, bool *enable* ) [inline], [static]

This function controls whether USB voltage regulator VLP standby write feature is enabled. Writing one to this bit allows the SOPT1[USBVSTBY] bit to be written. This register bit clears after a write to SOPT1[USBVSTBY].

Parameters

|                 |                                                                                                                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                |
| <i>enable</i>   | USB voltage regulator VLP standby write enable setting <ul style="list-style-type: none"> <li>• true: SOPT1[USBSSTBY] can be written.</li> <li>• false: SOPT1[USBSSTBY] cannot be written.</li> </ul> |

#### 40.2.6.79 static bool SIM\_HAL\_GetUsbVoltRegulatorInStdbyDuringVlprwCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the USB voltage regulator VLP standby write enable setting.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

enabled True if the USB voltage regulator VLP standby write is enabled.

### 40.2.6.80 static void SIM\_HAL\_SetUsbVoltRegulatorWriteCmd ( *uint32\_t baseAddr, bool enable* ) [inline], [static]

This function controls whether the USB voltage regulator write enable feature is enabled. Writing one to this bit allows the SOPT1[USBREGEN] bit to be written. This register bit clears after a write to SOTP1[USBREGEN].

Parameters

|                 |                                                                                                                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                        |
| <i>enable</i>   | USB voltage regulator enable write enable setting <ul style="list-style-type: none"><li>• true: SOPT1[USBSSTBY] can be written.</li><li>• false: SOPT1[USBSSTBY] cannot be written.</li></ul> |

### 40.2.6.81 static bool SIM\_HAL\_GetUsbVoltRegulatorWriteCmd ( *uint32\_t baseAddr* ) [inline], [static]

This function gets the USB voltage regulator enable write enable setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

enabled True if USB voltage regulator enable write is enabled.

### 40.2.6.82 static void CLOCK\_HAL\_SetOutDiv3 ( *uint32\_t baseAddr, uint8\_t setting* ) [inline], [static]

This function sets divide value OUTDIV3.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.83 static uint8\_t CLOCK\_HAL\_GetOutDiv3 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets divide value OUTDIV3.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current divide value.

#### 40.2.6.84 static void SIM\_HAL\_SetFlexbusSecurityLevelMode ( uint32\_t *baseAddr*, sim\_flexbus\_security\_level\_t *setting* ) [inline], [static]

This function sets the FlexBus security level setting. If the security is enabled, this field affects which CPU operations can access the off-chip via the FlexBus and DDR controller interfaces. This field has no effect if the security is not enabled.

Parameters

|                 |                                                                                                                                                                                                                                                                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                                                                                                                                                         |
| <i>setting</i>  | FlexBus security level setting <ul style="list-style-type: none"> <li>• 00: All off-chip accesses (op code and data) via the FlexBus and DDR controller are disallowed.</li> <li>• 10: Off-chip op code accesses are disallowed. Data accesses are allowed.</li> <li>• 11: Off-chip op code accesses and data accesses are allowed.</li> </ul> |

#### 40.2.6.85 static sim\_flexbus\_security\_level\_t SIM\_HAL\_GetFlexbusSecurityLevelMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the FlexBus security level setting.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting FlexBus security level setting

### 40.2.6.86 static void CLOCK\_HAL\_SetSdhcSrc ( *uint32\_t baseAddr, uint32\_t instance, clock\_sdhc\_src\_t setting* ) [inline], [static]

This function sets the SDHC clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

### 40.2.6.87 static *clock\_sdhc\_src\_t* CLOCK\_HAL\_GetSdhcSrc ( *uint32\_t baseAddr, uint32\_t instance* ) [inline], [static]

This function gets the SDHC clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

### 40.2.6.88 static void SIM\_HAL\_SetPtd7PadDriveStrengthMode ( *uint32\_t baseAddr, sim\_ptd7pad\_strenght\_t setting* ) [inline], [static]

This function controls the output drive strength of the PTD7 pin by selecting either one or two pads to drive it.

Parameters

|                 |                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                       |
| <i>setting</i>  | PTD7 pad drive strength setting <ul style="list-style-type: none"> <li>• 0: Single-pad drive strength for PTD7.</li> <li>• 1: Double pad drive strength for PTD7.</li> </ul> |

#### 40.2.6.89 static sim\_ptd7pad\_strenght\_t SIM\_HAL\_GetPtd7PadDriveStrengthMode ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the PTD7 pad drive strength setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting PTD7 pad drive strength setting

#### 40.2.6.90 static uint32\_t SIM\_HAL\_GetFlexnvmSize ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the FlexNVM size in the Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

size FlexNVM Size

#### 40.2.6.91 static uint32\_t SIM\_HAL\_GetEepromSize ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the EEPROM size in the Flash Configuration Register 1.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

size EEPROM Size

### 40.2.6.92 static uint32\_t SIM\_HAL\_GetFlexnvmPartition ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the FlexNVM partition in the Flash Configuration Register1

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

setting FlexNVM partition setting

### 40.2.6.93 static uint32\_t SIM\_HAL\_GetProgramFlashCmd ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the program flash maximum block 0 in Flash Configuration Register 1.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

status program flash status

### 40.2.6.94 static void CLOCK\_HAL\_SetTimeSrc ( uint32\_t *baseAddr*, uint32\_t *instance*, *clock\_time\_src\_t setting* ) [inline], [static]

This function sets the ethernet timestamp clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.95 static **clock\_time\_src\_t** CLOCK\_HAL\_GetTimeSrc ( **uint32\_t baseAddr, uint32\_t instance** ) [inline], [static]

This function gets the ethernet timestamp clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

#### 40.2.6.96 static void CLOCK\_HAL\_SetRmiiSrc ( **uint32\_t baseAddr, uint32\_t instance, clock\_rmii\_src\_t setting** ) [inline], [static]

This function sets the Ethernet RMII interface clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.97 static **clock\_rmii\_src\_t** CLOCK\_HAL\_GetRmiiSrc ( **uint32\_t baseAddr, uint32\_t instance** ) [inline], [static]

This function gets the Ethernet RMII interface clock source selection.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

### 40.2.6.98 static bool SIM\_HAL\_GetSwapProgramFlash ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Swap program flash flag in the Flash Configuration Register 2.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

status - Swap program flash flag(Active or Inactive)

### 40.2.6.99 void SIM\_HAL\_EnableDmaClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for DMA module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.100 void SIM\_HAL\_DisableDmaClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for DMA module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

**40.2.6.101 bool SIM\_HAL\_GetDmaGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function gets the clock gate state for DMA module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.102 void SIM\_HAL\_EnableDmamuxClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for DMAMUX module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.103 void SIM\_HAL\_DisableDmamuxClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for DMAMUX module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.104 bool SIM\_HAL\_GetDmamuxGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for DMAMUX module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for DMAMUX module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.105 void SIM\_HAL\_EnablePortClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for PORT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.106 void SIM\_HAL\_DisablePortClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for PORT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.107 bool SIM\_HAL\_GetPortGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for PORT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for PORT module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.108 void SIM\_HAL\_EnableMpuClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function enables the clock for MPU module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.109 void SIM\_HAL\_DisableMpuClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function disables the clock for MPU module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.110 bool SIM\_HAL\_GetMpuGateCmd ( *uint32\_t baseAddr, uint32\_t instance* )

This function will get the clock gate state for MPU module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

**40.2.6.111 void SIM\_HAL\_EnableEwmClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for EWM module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.112 void SIM\_HAL\_DisableEwmClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for EWM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.113 bool SIM\_HAL\_GetEwmGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for EWM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.114 void SIM\_HAL\_EnableFlexbusClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for FLEXBUS module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.115 void SIM\_HAL\_DisableFlexbusClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for FLEXBUS module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.116 **bool SIM\_HAL\_GetFlexbusGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function will get the clock gate state for FLEXBUS module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.117 **void SIM\_HAL\_EnableFtfClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for FTF module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.118 **void SIM\_HAL\_DisableFtfClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function disables the clock for FTF module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.119 **bool SIM\_HAL\_GetFtfGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function will get the clock gate state for FTF module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for FTF module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.120 void SIM\_HAL\_EnableCrcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for CRC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.121 void SIM\_HAL\_DisableCrcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for CRC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.122 bool SIM\_HAL\_GetCrcGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for CRC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.123 void SIM\_HAL\_EnableRngaClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function enables the clock for RNGA module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.124 void SIM\_HAL\_DisableRngaClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function disables the clock for RNGA module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.125 bool SIM\_HAL\_GetRngaGateCmd ( *uint32\_t baseAddr, uint32\_t instance* )

This function will get the clock gate state for RNGA module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.126 void SIM\_HAL\_EnableAdcClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function enables the clock for ADC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.127 void SIM\_HAL\_DisableAdcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for ADC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.128 bool SIM\_HAL\_GetAdcGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for ADC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for ADC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.129 void SIM\_HAL\_EnableCmpClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for CMP module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.130 void SIM\_HAL\_DisableCmpClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for CMP module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.131 bool SIM\_HAL\_GetCmpGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for CMP module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for CMP module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.132 void SIM\_HAL\_EnableDacClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for DAC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.133 void SIM\_HAL\_DisableDacClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for DAC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.134 bool SIM\_HAL\_GetDacGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for DAC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for DAC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.135 void SIM\_HAL\_EnableVrefClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for VREF module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.136 void SIM\_HAL\_DisableVrefClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for VREF module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.137 bool SIM\_HAL\_GetVrefGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for VREF module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.138 void SIM\_HAL\_EnableSaiClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for SAI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.139 void SIM\_HAL\_DisableSaiClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for SAI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.140 **bool SIM\_HAL\_GetSaiGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function will get the clock gate state for SAI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.141 **void SIM\_HAL\_EnablePdbClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for PDB module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.142 **void SIM\_HAL\_DisablePdbClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function disables the clock for PDB module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.143 **bool SIM\_HAL\_GetPdbGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function will get the clock gate state for PDB module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.144 void SIM\_HAL\_EnableFtmClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for FTM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.145 void SIM\_HAL\_DisableFtmClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for FTM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.146 bool SIM\_HAL\_GetFtmGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for FTM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

**40.2.6.147 void SIM\_HAL\_EnablePitClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for PIT module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.148 void SIM\_HAL\_DisablePitClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for PIT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.149 bool SIM\_HAL\_GetPitGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for PIT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for PIT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.150 void SIM\_HAL\_EnableLptimerClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for LPTIMER module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.151 void SIM\_HAL\_DisableLptimerClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for LPTIMER module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.152 bool SIM\_HAL\_GetLptimerGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for LPTIMER module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for LPTIMER module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.153 void SIM\_HAL\_EnableCmtClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for CMT module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.154 void SIM\_HAL\_DisableCmtClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for CMT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.155 bool SIM\_HAL\_GetCmtGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for CMT module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.156 void SIM\_HAL\_EnableRtcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for RTC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.157 void SIM\_HAL\_DisableRtcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for RTC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.158 bool SIM\_HAL\_GetRtcGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for RTC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for RTC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.159 void SIM\_HAL\_EnableEnetClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for ENET module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.160 void SIM\_HAL\_DisableEnetClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for ENET module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.161 bool SIM\_HAL\_GetEnetGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for ENET module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.162 void SIM\_HAL\_EnableUsbClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for USBFS module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.163 void SIM\_HAL\_DisableUsbClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for USBFS module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.164 bool SIM\_HAL\_GetUsbGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for USB module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for USB module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.165 void SIM\_HAL\_EnableUsbdcdClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for USBDCD module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.166 void SIM\_HAL\_DisableUsbdcdClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for USBDCD module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.167 bool SIM\_HAL\_GetUsbdcdGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for USBDCD module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### **40.2.6.168 void SIM\_HAL\_EnableFlexcanClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for FLEXCAN module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### **40.2.6.169 void SIM\_HAL\_DisableFlexcanClock ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function disables the clock for FLEXCAN module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### **40.2.6.170 bool SIM\_HAL\_GetFlexcanGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function will get the clock gate state for FLEXCAN module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

**40.2.6.171 void SIM\_HAL\_EnableSpiClock( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for SPI module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.172 void SIM\_HAL\_DisableSpiClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for SPI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.173 bool SIM\_HAL\_GetSpiGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for SPI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for SPI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.174 void SIM\_HAL\_EnableI2cClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for I2C module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.175 void SIM\_HAL\_DisableI2cClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for I2C module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.176 bool SIM\_HAL\_GetI2cGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for I2C module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for I2C module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.177 void SIM\_HAL\_EnableUartClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for UART module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.178 void SIM\_HAL\_DisableUartClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for UART module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.179 bool SIM\_HAL\_GetUartGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for UART module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for UART module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.180 void SIM\_HAL\_EnableSdhcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function enables the clock for SDHC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.181 void SIM\_HAL\_DisableSdhcClock ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function disables the clock for SDHC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

#### 40.2.6.182 bool SIM\_HAL\_GetSdhcGateCmd ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function will get the clock gate state for SDHC module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

#### 40.2.6.183 static void CLOCK\_HAL\_SetTpmSrc ( uint32\_t *baseAddr*, uint32\_t *instance*, clock TPM\_src\_t *setting* ) [inline], [static]

This function sets the clock selection of TPM.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

## SIM HAL driver

|                |                   |
|----------------|-------------------|
| <i>setting</i> | The value to set. |
|----------------|-------------------|

**40.2.6.184 static clock\_tpm\_src\_t CLOCK\_HAL\_GetTpmSrc ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets the clock selection of TPM.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

**40.2.6.185 static void CLOCK\_HAL\_SetCopSrc ( uint32\_t *baseAddr*, clock\_cop\_src\_t *setting* ) [inline], [static]**

This function sets the clock selection of COP.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

**40.2.6.186 static clock\_cop\_src\_t CLOCK\_HAL\_GetCopSrc ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the clock selection of COP.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current selection.

**40.2.6.187 static void SIM\_HAL\_SetLpUartRxSrcMode( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_uart\_rxsrc\_t *select* ) [inline], [static]**

This function selects the source for the LPUARTx receive data.

## SIM HAL driver

Parameters

|                 |                                                                                                                                                           |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Register base address of SIM.                                                                                                                             |
| <i>instance</i> | LPUART instance.                                                                                                                                          |
| <i>select</i>   | the source for the LPUARTx receive data <ul style="list-style-type: none"><li>• 00: LPUARTx_RX pin.</li><li>• 01: CMP0.</li><li>• 11: Reserved.</li></ul> |

**40.2.6.188 static sim\_uart\_rxsrc\_t SIM\_HAL\_GetLpUartRxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets the LPUARTx receive data source select setting.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>baseAddr</i> | Register base address of SIM. |
| <i>instance</i> | LPUART instance.              |

Returns

select UARTx receive data source select setting

**40.2.6.189 static void SIM\_HAL\_SetLpUartTxSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_uart\_txsrc\_t *select* ) [inline], [static]**

This function selects the source for the LPUARTx transmit data.

Parameters

|                 |                                                                                                                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Register base address of SIM.                                                                                                                                                                           |
| <i>instance</i> | LPUART instance.                                                                                                                                                                                        |
| <i>select</i>   | the source for the UARTx transmit data <ul style="list-style-type: none"><li>• 00: LPUARTx_TX pin.</li><li>• 01: LPUARTx_TX pin modulated with FTM1 channel 0 output.</li><li>• 11: Reserved.</li></ul> |

**40.2.6.190 static sim\_uart\_txsrc\_t SIM\_HAL\_GetLpUartTxSrcMode ( uint32\_t *baseAddr*,  
                  uint32\_t *instance* ) [inline], [static]**

This function gets the LPUARTx transmit data source select setting.

## SIM HAL driver

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>baseAddr</i> | Register base address of SIM. |
| <i>instance</i> | LPUART instance.              |

Returns

select UARTx transmit data source select setting

**40.2.6.191 static void SIM\_HAL\_SetLpUartOpenDrainCmd ( uint32\_t *baseAddr*, uint32\_t *instance*, bool *enable* ) [inline], [static]**

This function enables/disables the LPUARTx Open Drain.

Parameters

|                 |                                                                                                                                                                 |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Register base address of SIM.                                                                                                                                   |
| <i>instance</i> | LPUART instance.                                                                                                                                                |
| <i>enable</i>   | Enable/disable LPUARTx Open Drain <ul style="list-style-type: none"><li>• True: Enable LPUARTx Open Drain</li><li>• False: Disable LPUARTx Open Drain</li></ul> |

**40.2.6.192 static bool SIM\_HAL\_GetLpUartOpenDrainCmd ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets the LPUARTx Open Drain Enable setting.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>baseAddr</i> | Register base address of SIM. |
| <i>instance</i> | LPUART instance.              |

Returns

enabled True if LPUARTx Open Drain is enabled.

**40.2.6.193 static void SIM\_HAL\_SetTpmChSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, uint8\_t *channel*, sim\_tpm\_ch\_src\_t *select* ) [inline], [static]**

This function selects the Timer/PWM x channel y input capture source.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.     |
| <i>instance</i> | device instance.                           |
| <i>channel</i>  | TPM channel y                              |
| <i>select</i>   | Timer/PWM x channel y input capture source |

#### 40.2.6.194 static sim\_tpm\_ch\_src\_t SIM\_HAL\_GetTpmChSrcMode ( uint32\_t *baseAddr*, uint32\_t *instance*, uint8\_t *channel* ) [inline], [static]

This function gets the Timer/PWM x channel y input capture source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>channel</i>  | Tpm channel y                          |

Returns

select Timer/PWM x channel y input capture source select setting

#### 40.2.6.195 void SIM\_HAL\_SetTpmExternalClkPinSelMode ( uint32\_t *baseAddr*, uint32\_t *instance*, sim\_tpm\_clk\_sel\_t *select* )

This function selects the source of the Timer/PWM x external clock pin select.

Parameters

|                 |                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                                       |
| <i>instance</i> | device instance.                                                                                                                                                                                                             |
| <i>select</i>   | Timer/PWM x external clock pin select <ul style="list-style-type: none"> <li>• 0: Timer/PWM x external clock driven by the TPM_CLKIN0 pin.</li> <li>• 1: Timer/PWM x external clock driven by the TPM_CLKIN1 pin.</li> </ul> |

#### 40.2.6.196 sim\_tpm\_clk\_sel\_t SIM\_HAL\_GetTpmExternalClkPinSelMode ( uint32\_t *baseAddr*, uint32\_t *instance* )

This function gets the Timer/PWM x external clock pin select setting.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select Timer/PWM x external clock pin select setting

### 40.2.6.197 void SIM\_HAL\_EnableTpmClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function enables the clock for TPM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.198 void SIM\_HAL\_DisableTpmClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function disables the clock for TPM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.199 bool SIM\_HAL\_GetTpmGateCmd ( *uint32\_t baseAddr, uint32\_t instance* )

This function gets the clock gate state for TPM module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

**40.2.6.200 void SIM\_HAL\_EnableTsiClock( uint32\_t *baseAddr*, uint32\_t *instance* )**

This function enables the clock for TSI module.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.201 void SIM\_HAL\_DisableTsiClock ( *uint32\_t baseAddr, uint32\_t instance* )

This function disables the clock for TSI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

### 40.2.6.202 bool SIM\_HAL\_GetTsiGateCmd ( *uint32\_t baseAddr, uint32\_t instance* )

This function gets the clock gate state for TSI module.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | module device instance                 |

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 40.2.6.203 static void CLOCK\_HAL\_SetLpsciSrc ( *uint32\_t baseAddr, uint32\_t instance, clock\_lpisci\_src\_t setting* ) [inline], [static]

This function sets the LPSCI clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

|                |                   |
|----------------|-------------------|
| <i>setting</i> | The value to set. |
|----------------|-------------------|

**40.2.6.204 static clock\_lpisci\_src\_t CLOCK\_HAL\_GetLpisciSrc ( uint32\_t *baseAddr*, uint32\_t *instance* ) [inline], [static]**

This function gets the LPSCI clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

**40.2.6.205 static uint32\_t SIM\_HAL\_GetSramSize ( uint32\_t *baseAddr* ) [inline], [static]**

This function gets the Kinetis SramSize in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis SramSize

**40.2.6.206 static void CLOCK\_HAL\_SetOutDiv5ENCmd ( uint32\_t *baseAddr*, bool *setting* ) [inline], [static]**

This function sets divide value OUTDIV5EN.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

## SIM HAL driver

**40.2.6.207 static bool CLOCK\_HAL\_GetOutDiv5ENCmd ( uint32\_t *baseAddr* )  
[inline], [static]**

This function gets divide value OUTDIV5EN.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current divide value.

#### 40.2.6.208 static void CLOCK\_HAL\_SetOutDiv5 ( uint32\_t *baseAddr*, uint8\_t *setting* ) [inline], [static]

This function sets divide value OUTDIV5.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>setting</i>  | The value to set.                      |

#### 40.2.6.209 static uint8\_t CLOCK\_HAL\_GetOutDiv5 ( uint32\_t *baseAddr* ) [inline], [static]

This function gets divide value OUTDIV5.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Current divide value.

#### 40.2.6.210 void CLOCK\_HAL\_SetAdcAltClkSrc ( uint32\_t *baseAddr*, uint32\_t *instance*, *clock\_adc\_alt\_src\_t adcAltSrcSel* )

This function sets the ADC ALT clock source selection setting.

Parameters

## SIM HAL driver

|                     |                                        |
|---------------------|----------------------------------------|
| <i>baseAddr</i>     | Base address for current SIM instance. |
| <i>instance</i>     | ADC module instance.                   |
| <i>adcAltSrcSel</i> | ADC ALT clock source.                  |

**40.2.6.211 `clock_adc_alt_src_t CLOCK_HAL_GetAdcAltClkSrc ( uint32_t baseAddr, uint32_t instance )`**

This function gets the ADC ALT clock source selection setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | ADC module instance.                   |

Returns

the ADC ALT clock source selection.

**40.2.6.212 `void SIM_HAL_SetUartOpenDrainMode ( uint32_t baseAddr, uint32_t instance, bool enable )`**

This function enables/disables open drain for UARTx.

Parameters

|                 |                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                              |
| <i>instance</i> | device instance.                                                                                                                                    |
| <i>enable</i>   | Enable open drain for UARTx <ul style="list-style-type: none"><li>• true: Open drain is enabled.</li><li>• false: Open drain is disabled.</li></ul> |

**40.2.6.213 `bool SIM_HAL_GetUartOpenDrainMode ( uint32_t baseAddr, uint32_t instance )`**

This function Gets the UARTx open drain enable setting for UARTx.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

#### 40.2.6.214 static void CLOCK\_HAL\_SetFtmFixFreqClkSrc ( uint32\_t *baseAddr*, clock\_ftm\_fixedfreq\_src\_t *ftmFixedFreqSel* ) [inline], [static]

This function sets the FTM Fixed clock source selection setting.

Parameters

|                        |                                        |
|------------------------|----------------------------------------|
| <i>baseAddr</i>        | Base address for current SIM instance. |
| <i>ftmFixedFreqSel</i> | FTM Fixed clock source.                |

#### 40.2.6.215 static clock\_ftm\_fixedfreq\_src\_t CLOCK\_HAL\_GetFtmFixFreqClkSrc ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the FTM Fixed clock source selection setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

the FTM Fixed clock source selection.

#### 40.2.6.216 static void SIM\_HAL\_SetFtmCarrierFreqMode ( uint32\_t *baseAddr*, sim\_ftm\_ftl\_carrier\_sel\_t *select* ) [inline], [static]

This function Sets the Carrier frequency selection for FTM0/2 output channel.

Parameters

|                 |                                                                                                                                                                                                         |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance.                                                                                                                                                                  |
| <i>select</i>   | Carrier frequency source select. <ul style="list-style-type: none"><li>• 0 : FTM1_CH1 output provides the carrier signal;</li><li>• 1 : LPTMR0 pre-scaler output provides the carrier signal;</li></ul> |

## SIM HAL driver

**40.2.6.217 static sim\_ftm\_flt\_carrier\_sel\_t SIM\_HAL\_GetFtmCarrierFreqMode ( uint32\_t baseAddr ) [inline], [static]**

This function gets Carrier frequency selection setting for FTM0/2 output channel.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

Carrier frequency selection;

#### 40.2.6.218 static uint32\_t SIM\_HAL\_GetSubFamId ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Kinetis SubFam ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis SubFam ID

#### 40.2.6.219 static uint32\_t SIM\_HAL\_GetSramSizeld ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Kinetis SRAMSIZE ID in System Device ID register.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

id Kinetis SRAMSIZE ID

#### 40.2.6.220 static uint32\_t SIM\_HAL\_GetFlashMaxAddrBlock ( uint32\_t *baseAddr* ) [inline], [static]

This function gets the Flash maximum block in Flash Configuration Register 2.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Base address for current SIM instance. |
|-----------------|----------------------------------------|

Returns

address Flash maximum block address

## 40.2.7 K02F12810 SIM HAL driver

### 40.2.7.1 Overview

The section describes the enumerations, macros and data structures for K02F12810 SIM HAL driver.

#### Files

- file `fsl_sim_hal_MK02F12810.h`

#### Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*

#### Enumerations

- enum `clock_wdog_src_k02f12810_t` {
   
`kClockWdogSrcLpoClk,`
  
`kClockWdogSrcAltClk }`
  
*WDOG clock source select.*
- enum `clock_trace_src_k02f12810_t` {
   
`kClockTraceSrcMcgoutClk,`
  
`kClockTraceSrcCoreClk }`
  
*Debug trace clock source select.*
- enum `clock_port_filter_src_k02f12810_t` {
   
`kClockPortFilterSrcBusClk,`
  
`kClockPortFilterSrcLpoClk }`
  
*PORTx digital input filter clock source select.*
- enum `clock_lptmr_src_k02f12810_t` {
   
`kClockLptmrSrcMcgIrClk,`
  
`kClockLptmrSrcLpoClk,`
  
`kClockLptmrSrcEr32kClk,`
  
`kClockLptmrSrcOsc0erClkUndiv }`
  
*LPTMR clock source select.*
- enum `clock_pllfl_sel_k02f12810_t` {
   
`kClockPliflSelFll = 0U,`
  
`kClockPliflSelIrc48M = 3U }`
  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k02f12810_t` {
   
`kClockEr32kSrcOsc0 = 0U,`
  
`kClockEr32kSrcLpo = 3U }`
  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k02f12810_t` {

## SIM HAL driver

- ```
kClockClkoutSelFlashClk = 2U,  
kClockClkoutSelLpoClk = 3U,  
kClockClkoutSelMcgIrClk = 4U,  
kClockClkoutSelOsc0erClk = 6U,  
kClockClkoutSelIrc48M = 7U }  
    SIM CLKOUT_SEL clock source select.  
• enum clock\_osc32kout\_sel\_k02f12810\_t {  
    kClockOsc32koutNone = 0U,  
    kClockOsc32koutPte0 = 1U,  
    kClockOsc32koutPte26 = 2U }  
    SIM OSC32KOUT selection.  
• enum sim\_adc\_pretrg\_sel\_k02f12810\_t {  
    kSimAdcPretrgselA,  
    kSimAdcPretrgselB }  
    SIM ADCx pre-trigger select.  
• enum sim\_adc\_trg\_sel\_k02f12810\_t {  
    kSimAdcTrgSelExt = 0U,  
    kSimAdcTrgSelHighSpeedComp0 = 1U,  
    kSimAdcTrgSelHighSpeedComp1 = 2U,  
    kSimAdcTrgSelPit0 = 4U,  
    kSimAdcTrgSelPit1 = 5U,  
    kSimAdcTrgSelPit2 = 6U,  
    kSimAdcTrgSelPit3 = 7U,  
    kSimAdcTrgSelFtm0 = 8U,  
    kSimAdcTrgSelFtm1 = 9U,  
    kSimAdcTrgSelFtm2 = 10U,  
    kSimAdcTrgSelLptimer = 14U }  
    SIM ADCx trigger select.  
• enum sim\_uart\_rxsrcc\_k02f12810\_t {  
    kSimUartRxsrccPin,  
    kSimUartRxsrccCmp0,  
    kSimUartRxsrccCmp1 }  
    SIM receive data source select.  
• enum sim\_uart\_txsrc\_k02f12810\_t {  
    kSimUartTxsrcPin,  
    kSimUartTxsrcFtm1,  
    kSimUartTxsrcFtm2 }  
    SIM transmit data source select.  
• enum sim\_ftm\_trg\_src\_k02f12810\_t {  
    kSimFtmTrgSrc0,  
    kSimFtmTrgSrc1 }  
    SIM FlexTimer x trigger y select.  
• enum sim\_ftm\_clk\_sel\_k02f12810\_t {  
    kSimFtmClkSel0,  
    kSimFtmClkSel1 }  
    SIM FlexTimer external clock select.  
• enum sim\_ftm\_ch\_src\_k02f12810\_t {
```

```
kSimFtmChSrc0,
kSimFtmChSrc1,
kSimFtmChSrc2,
kSimFtmChSrc3 }
```

SIM FlexTimer x channel y input capture source select.

- enum [sim_ftm_ch_out_src_k02f12810_t](#) {
 kSimFtmChOutSrc0,
 kSimFtmChOutSrc1 }

SIM FlexTimer x channel y output source select.

- enum [sim_ftm_flt_sel_k02f12810_t](#) {
 kSimFtmFltSel0,
 kSimFtmFltSel1 }

SIM FlexTimer x Fault y select.

- enum [sim_tpm_clk_sel_k02f12810_t](#) {
 kSimTpmClkSel0,
 kSimTpmClkSel1 }

SIM Timer/PWM external clock select.

- enum [sim_tpm_ch_src_k02f12810_t](#) {
 kSimTpmChSrc0,
 kSimTpmChSrc1 }

SIM Timer/PWM x channel y input capture source select.

- enum [sim_clock_gate_name_k02f12810_t](#)

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

40.2.7.2 Macro Definition Documentation

40.2.7.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.7.3 Enumeration Type Documentation

40.2.7.3.1 enum [clock_wdog_src_k02f12810_t](#)

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

40.2.7.3.2 enum [clock_trace_src_k02f12810_t](#)

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

SIM HAL driver

40.2.7.3.3 enum clock_port_filter_src_k02f12810_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.7.3.4 enum clock_lptmr_src_k02f12810_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

kClockLptmrSrcLpoClk LPO clock.

kClockLptmrSrcEr32kClk ERCLK32K clock.

kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

40.2.7.3.5 enum clock_pllfl_sel_k02f12810_t

Enumerator

kClockPllFlSelFll Fll clock.

kClockPllFlSelIrc48M IRC48MCLK.

40.2.7.3.6 enum clock_er32k_src_k02f12810_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).

kClockEr32kSrcLpo LPO clock.

40.2.7.3.7 enum clock_clkout_src_k02f12810_t

Enumerator

kClockClkoutSelFlashClk Flash clock.

kClockClkoutSelLpoClk LPO clock.

kClockClkoutSelMcgIrClk MCGIRCLK.

kClockClkoutSelOsc0erClk OSC0ERCLK.

kClockClkoutSelIrc48M IRC48MCLK.

40.2.7.3.8 enum clock_osc32kout_sel_k02f12810_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

40.2.7.3.9 enum sim_adc_pretrg_sel_k02f12810_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

40.2.7.3.10 enum sim_adc_trg_sel_k02f12810_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

40.2.7.3.11 enum sim_uart_rxsra_k02f12810_t

Enumerator

- kSimUartRxsraPin* UARTx_RX Pin.
- kSimUartRxsraCmp0* CMP0.
- kSimUartRxsraCmp1* CMP1.

40.2.7.3.12 enum sim_uart_txsrc_k02f12810_t

Enumerator

- kSimUartTxsrcPin* UARTx_TX Pin.

SIM HAL driver

kSimUartTxsrcFtm1 UARTx_TX pin modulated with FTM1 channel 0 output.

kSimUartTxsrcFtm2 UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.7.3.13 enum sim_ftm_trg_src_k02f12810_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.

kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

40.2.7.3.14 enum sim_ftm_clk_sel_k02f12810_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.

kSimFtmClkSel1 FTM CLKIN1 pin.

40.2.7.3.15 enum sim_ftm_ch_src_k02f12810_t

Enumerator

kSimFtmChSrc0 FlexTimer x channel y input capture source 0.

kSimFtmChSrc1 FlexTimer x channel y input capture source 1.

kSimFtmChSrc2 FlexTimer x channel y input capture source 2.

kSimFtmChSrc3 FlexTimer x channel y input capture source 3.

40.2.7.3.16 enum sim_ftm_ch_out_src_k02f12810_t

Enumerator

kSimFtmChOutSrc0 FlexTimer x channel y output source selection 0.

kSimFtmChOutSrc1 FlexTimer x channel y output source selection 1.

40.2.7.3.17 enum sim_ftm_flt_sel_k02f12810_t

Enumerator

kSimFtmFltSel0 FlexTimer x fault y select 0.

kSimFtmFltSel1 FlexTimer x fault y select 1.

40.2.7.3.18 enum sim_tpm_clk_sel_k02f12810_t

Enumerator

kSimTpmClkSel0 Timer/PWM TPM_CLKIN0 pin.

kSimTpmClkSel1 Timer/PWM TPM_CLKIN1 pin.

40.2.7.3.19 enum sim_tpm_ch_src_k02f12810_t

Enumerator

kSimTpmChSrc0 TPMx_CH0 signal.

kSimTpmChSrc1 CMP0 output.

40.2.7.3.20 enum sim_clock_gate_name_k02f12810_t

SIM HAL driver

40.2.8 K22F12810 SIM HAL driver

40.2.8.1 Overview

The section describes the enumerations, macros and data structures for K22F12810 SIM HAL driver.

Macros

- #define `FSL_SIM_SCGC_BIT(SCGCx, n)` (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum `clock_wdog_src_k22f12810_t`{
 `kClockWdogSrcLpoClk,`
 `kClockWdogSrcAltClk }`
WDOG clock source select.
- enum `clock_trace_src_k22f12810_t`{
 `kClockTraceSrcMcgoutClk,`
 `kClockTraceSrcCoreClk }`
Debug trace clock source select.
- enum `clock_port_filter_src_k22f12810_t`{
 `kClockPortFilterSrcBusClk,`
 `kClockPortFilterSrcLpoClk }`
PORTx digital input filter clock source select.
- enum `clock_lptmr_src_k22f12810_t`{
 `kClockLptmrSrcMcgIrClk,`
 `kClockLptmrSrcLpoClk,`
 `kClockLptmrSrcEr32kClk,`
 `kClockLptmrSrcOsc0erClkUndiv }`
LPTMR clock source select.
- enum `clock_usbfs_src_k22f12810_t`{
 `kClockUsbfsSrcExt,`
 `kClockUsbfsSrcPllFllSel }`
SIM USB FS clock source.
- enum `clock_lpuart_src_k22f12810_t`{
 `kClockLpuartSrcNone,`
 `kClockLpuartSrcPllFllSel,`
 `kClockLpuartSrcOsc0erClk,`
 `kClockLpuartSrcMcgIrClk }`
SIM LPUART clock source.
- enum `clock_sai_src_k22f12810_t`{
 `kClockSaiSrcSysClk = 0U,`
 `kClockSaiSrcOsc0erClk = 1U,`
 `kClockSaiSrcPllFllSel = 3U }`
SAI clock source.

- enum `clock_pllfl_sel_k22f12810_t` {

 `kClockPliflSelFll` = 0U,

 `kClockPliflSelIrc48M` = 3U }

 SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k22f12810_t` {

 `kClockEr32kSrcOsc0` = 0U,

 `kClockEr32kSrcRtc` = 2U,

 `kClockEr32kSrcLpo` = 3U }

 SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k22f12810_t` {

 `kClockClkoutSelFlashClk` = 2U,

 `kClockClkoutSelLpoClk` = 3U,

 `kClockClkoutSelMcgIrClk` = 4U,

 `kClockClkoutSelRtc` = 5U,

 `kClockClkoutSelOsc0erClk` = 6U,

 `kClockClkoutSelIrc48M` = 7U }

 SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k22f12810_t` {

 `kClockRtcoutSrc1Hz`,

 `kClockRtcoutSrc32kHz` }

 SIM RTCCLKOUTSEL clock source select.
- enum `clock_osc32kout_sel_k22f12810_t` {

 `kClockOsc32koutNone` = 0U,

 `kClockOsc32koutPte0` = 1U,

 `kClockOsc32koutPte26` = 2U }

 SIM OSC32KOUT selection.
- enum `sim_usbsstby_mode_k22f12810_t` {

 `kSimUsbsstbyNoRegulator`,

 `kSimUsbsstbyWithRegulator` }

 SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k22f12810_t` {

 `kSimUsbvstbyNoRegulator`,

 `kSimUsbvstbyWithRegulator` }

 SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
- enum `sim_adc_pretrg_sel_k22f12810_t` {

 `kSimAdcPretrgselA`,

 `kSimAdcPretrgselB` }

 SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_k22f12810_t` {

SIM HAL driver

```
kSimAdcTrgSelExt = 0U,  
kSimAdcTrgSelHighSpeedComp0 = 1U,  
kSimAdcTrgSelHighSpeedComp1 = 2U,  
kSimAdcTrgSelPit0 = 4U,  
kSimAdcTrgSelPit1 = 5U,  
kSimAdcTrgSelPit2 = 6U,  
kSimAdcTrgSelPit3 = 7U,  
kSimAdcTrgSelFtm0 = 8U,  
kSimAdcTrgSelFtm1 = 9U,  
kSimAdcTrgSelFtm2 = 10U,  
kSimAdcTrgSelRtcAlarm = 12U,  
kSimAdcTrgSelRtcSec = 13U,  
kSimAdcTrgSelLptimer = 14U }
```

SIM ADCx trigger select.

- enum `sim_lpuart_rxsrck22f12810_t` {
 kSimLpuartRxsrPin,
 kSimLpuartRxsrCmp0,
 kSimLpuartRxsrCmp1 }

SIM LPUART RX source.

- enum `sim_uart_rxsrck22f12810_t` {
 kSimUartRxsrPin,
 kSimUartRxsrCmp0,
 kSimUartRxsrCmp1 }

SIM receive data source select.

- enum `sim_uart_txsrc_k22f12810_t` {
 kSimUartTxsrPin,
 kSimUartTxsrcFtm1,
 kSimUartTxsrcFtm2 }

SIM transmit data source select.

- enum `sim_ftm_trg_src_k22f12810_t` {
 kSimFtmTrgSrc0,
 kSimFtmTrgSrc1 }

SIM FlexTimer x trigger y select.

- enum `sim_ftm_clk_sel_k22f12810_t` {
 kSimFtmClkSel0,
 kSimFtmClkSel1 }

SIM FlexTimer external clock select.

- enum `sim_ftm_ch_src_k22f12810_t` {
 kSimFtmChSrc0,
 kSimFtmChSrc1,
 kSimFtmChSrc2,
 kSimFtmChSrc3 }

SIM FlexTimer x channel y input capture source select.

- enum `sim_ftm_ch_out_src_k22f12810_t` {
 kSimFtmChOutSrc0,
 kSimFtmChOutSrc1 }

- *SIM FlexTimer x channel y output source select.*
• enum `sim_ftm_ftl_sel_k22f12810_t` {
`kSimFtmFltSel0,`
`kSimFtmFltSel1 }`
- *SIM FlexTimer x Fault y select.*
• enum `sim_tpm_clk_sel_k22f12810_t` {
`kSimTpmClkSel0,`
`kSimTpmClkSel1 }`
- *SIM Timer/PWM external clock select.*
• enum `sim_tpm_ch_src_k22f12810_t` {
`kSimTpmChSrc0,`
`kSimTpmChSrc1 }`
- *SIM Timer/PWM x channel y input capture source select.*
• enum `sim_cmtuartpad_strenght_k22f12810_t` {
`kSimCmtuartSinglePad,`
`kSimCmtuartDualPad }`
- *SIM CMT/UART pad drive strength.*
• enum `sim_ptd7pad_strenght_k22f12810_t` {
`kSimPtd7padSinglePad,`
`kSimPtd7padDualPad }`
- *SIM PTD7 pad drive strength.*
• enum `sim_flexport_security_level_k22f12810_t` {
`kSimFbslLevel0,`
`kSimFbslLevel1,`
`kSimFbslLevel2,`
`kSimFbslLevel3 }`
- *SIM FlexBus security level.*
• enum `sim_clock_gate_name_k22f12810_t`
Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

40.2.8.2 Macro Definition Documentation

40.2.8.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.8.3 Enumeration Type Documentation

40.2.8.3.1 enum clock_wdog_src_k22f12810_t

Enumerator

`kClockWdogSrcLpoClk` LPO.

`kClockWdogSrcAltClk` Alternative clock, for this SOC it is Bus clock.

SIM HAL driver

40.2.8.3.2 enum clock_trace_src_k22f12810_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.8.3.3 enum clock_port_filter_src_k22f12810_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.8.3.4 enum clock_lptmr_src_k22f12810_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

kClockLptmrSrcLpoClk LPO clock.

kClockLptmrSrcEr32kClk ERCLK32K clock.

kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

40.2.8.3.5 enum clock_usbfs_src_k22f12810_t

Enumerator

kClockUsbfsSrcExt External bypass clock (USB_CLKIN)

kClockUsbfsSrcPllFllSel Clock divider USB FS clock.

40.2.8.3.6 enum clock_lpuart_src_k22f12810_t

Enumerator

kClockLpuartSrcNone Clock disabled.

kClockLpuartSrcPllFllSel Clock as selected by SOPT2[PLLFLSEL].

kClockLpuartSrcOsc0erClk OSCERCLK.

kClockLpuartSrcMcgIrClk MCGIRCLK.

40.2.8.3.7 enum clock_sai_src_k22f12810_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllFllSel MCGPLLCLK.

40.2.8.3.8 enum clock_pllfl_sel_k22f12810_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelIrc48M IRC48MCLK.

40.2.8.3.9 enum clock_er32k_src_k22f12810_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

40.2.8.3.10 enum clock_clkout_src_k22f12810_t

Enumerator

kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelRtc RTC 32k clock.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

40.2.8.3.11 enum clock_rtcout_src_k22f12810_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

SIM HAL driver

40.2.8.3.12 enum clock_osc32kout_sel_k22f12810_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

40.2.8.3.13 enum sim_usbsstby_mode_k22f12810_t

Enumerator

- kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes
- kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

40.2.8.3.14 enum sim_usbvstby_mode_k22f12810_t

Enumerator

- kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes
- kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

40.2.8.3.15 enum sim_adc_pretrg_sel_k22f12810_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

40.2.8.3.16 enum sim_adc_trg_sel_k22f12810_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.

kSimAdcTrgSelRtcAlarm RTC alarm.
kSimAdcTrgSelRtcSec RTC seconds.
kSimAdcTrgSelLptimer Low-power timer trigger.

40.2.8.3.17 enum sim_lpuart_rxsrv_k22f12810_t

Enumerator

kSimLpuartRxsrvPin LPUARTx_RX Pin.
kSimLpuartRxsrvCmp0 CMP0.
kSimLpuartRxsrvCmp1 CMP1.

40.2.8.3.18 enum sim_uart_rxsrv_k22f12810_t

Enumerator

kSimUartRxsrvPin UARTx_RX Pin.
kSimUartRxsrvCmp0 CMP0.
kSimUartRxsrvCmp1 CMP1.

40.2.8.3.19 enum sim_uart_txsrc_k22f12810_t

Enumerator

kSimUartTxsrcPin UARTx_TX Pin.
kSimUartTxsrcFtm1 UARTx_TX pin modulated with FTM1 channel 0 output.
kSimUartTxsrcFtm2 UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.8.3.20 enum sim_ftm_trg_src_k22f12810_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

40.2.8.3.21 enum sim_ftm_clk_sel_k22f12810_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

SIM HAL driver

40.2.8.3.22 enum sim_ftm_ch_src_k22f12810_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

40.2.8.3.23 enum sim_ftm_ch_out_src_k22f12810_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source selection 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source selection 1.

40.2.8.3.24 enum sim_ftm_flt_sel_k22f12810_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

40.2.8.3.25 enum sim_tpm_clk_sel_k22f12810_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

40.2.8.3.26 enum sim_tpm_ch_src_k22f12810_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

40.2.8.3.27 enum sim_cmtuartpad_strenght_k22f12810_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

40.2.8.3.28 enum sim_ptd7pad_strength_k22f12810_t

Enumerator

kSimPtd7padSinglePad Single-pad drive strength for PTD7.

kSimPtd7padDualPad Dual-pad drive strength for PTD7.

40.2.8.3.29 enum sim_flexbus_security_level_k22f12810_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

40.2.8.3.30 enum sim_clock_gate_name_k22f12810_t

40.2.9 K22F25612 SIM HAL driver

40.2.9.1 Overview

The section describes the enumerations, macros and data structures for K22F25612 SIM HAL driver.

Macros

- #define `FSL_SIM_SCGC_BIT(SCGCx, n)` (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum `clock_wdog_src_k22f25612_t`{
 `kClockWdogSrcLpoClk,`
 `kClockWdogSrcAltClk }`
WDOG clock source select.
- enum `clock_trace_src_k22f25612_t`{
 `kClockTraceSrcMcgoutClk,`
 `kClockTraceSrcCoreClk }`
Debug trace clock source select.
- enum `clock_port_filter_src_k22f25612_t`{
 `kClockPortFilterSrcBusClk,`
 `kClockPortFilterSrcLpoClk }`
PORTx digital input filter clock source select.
- enum `clock_lptmr_src_k22f25612_t`{
 `kClockLptmrSrcMcgIrClk,`
 `kClockLptmrSrcLpoClk,`
 `kClockLptmrSrcEr32kClk,`
 `kClockLptmrSrcOsc0erClkUndiv }`
LPTMR clock source select.
- enum `clock_usbfs_src_k22f25612_t`{
 `kClockUsbfsSrcExt,`
 `kClockUsbfsSrcPllFllSel }`
SIM USB FS clock source.
- enum `clock_lpuart_src_k22f25612_t`{
 `kClockLpuartSrcNone,`
 `kClockLpuartSrcPllFllSel,`
 `kClockLpuartSrcOsc0erClk,`
 `kClockLpuartSrcMcgIrClk }`
SIM LPUART clock source.
- enum `clock_sai_src_k22f25612_t`{
 `kClockSaiSrcSysClk = 0U,`
 `kClockSaiSrcOsc0erClk = 1U,`
 `kClockSaiSrcPllFllSel = 3U }`
SAI clock source.

- enum `clock_pllfl_sel_k22f25612_t` {

 `kClockPliflSelFll` = 0U,

 `kClockPliflSelPll` = 1U,

 `kClockPliflSelIrc48M` = 3U }

 SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k22f25612_t` {

 `kClockEr32kSrcOsc0` = 0U,

 `kClockEr32kSrcRtc` = 2U,

 `kClockEr32kSrcLpo` = 3U }

 SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k22f25612_t` {

 `kClockClkoutSelFlashClk` = 2U,

 `kClockClkoutSelLpoClk` = 3U,

 `kClockClkoutSelMcgIrClk` = 4U,

 `kClockClkoutSelRtc` = 5U,

 `kClockClkoutSelOsc0erClk` = 6U,

 `kClockClkoutSelIrc48M` = 7U }

 SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k22f25612_t` {

 `kClockRtcoutSrc1Hz`,

 `kClockRtcoutSrc32kHz` }

 SIM RTCCLKOUTSEL clock source select.
- enum `clock_osc32kout_sel_k22f25612_t` {

 `kClockOsc32koutNone` = 0U,

 `kClockOsc32koutPte0` = 1U,

 `kClockOsc32koutPte26` = 2U }

 SIM OSC32KOUT selection.
- enum `sim_usbsstby_mode_k22f25612_t` {

 `kSimUsbstbyNoRegulator`,

 `kSimUsbstbyWithRegulator` }

 SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k22f25612_t` {

 `kSimUsbvstbyNoRegulator`,

 `kSimUsbvstbyWithRegulator` }

 SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
- enum `sim_adc_pretrg_sel_k22f25612_t` {

 `kSimAdcPretrgselA`,

 `kSimAdcPretrgselB` }

 SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_k22f25612_t` {

SIM HAL driver

```
kSimAdcTrgSelExt = 0U,  
kSimAdcTrgSelHighSpeedComp0 = 1U,  
kSimAdcTrgSelHighSpeedComp1 = 2U,  
kSimAdcTrgSelPit0 = 4U,  
kSimAdcTrgSelPit1 = 5U,  
kSimAdcTrgSelPit2 = 6U,  
kSimAdcTrgSelPit3 = 7U,  
kSimAdcTrgSelFtm0 = 8U,  
kSimAdcTrgSelFtm1 = 9U,  
kSimAdcTrgSelFtm2 = 10U,  
kSimAdcTrgSelRtcAlarm = 12U,  
kSimAdcTrgSelRtcSec = 13U,  
kSimAdcTrgSelLptimer = 14U }
```

SIM ADCx trigger select.

- enum `sim_lpuart_rxsrck22f25612_t` {
 kSimLpuartRxsrPin,
 kSimLpuartRxsrCmp0,
 kSimLpuartRxsrCmp1 }

SIM LPUART RX source.

- enum `sim_uart_rxsrck22f25612_t` {
 kSimUartRxsrPin,
 kSimUartRxsrCmp0,
 kSimUartRxsrCmp1 }

SIM receive data source select.

- enum `sim_uart_txsrc_k22f25612_t` {
 kSimUartTxsrPin,
 kSimUartTxsrcFtm1,
 kSimUartTxsrcFtm2 }

SIM transmit data source select.

- enum `sim_ftm_trg_src_k22f25612_t` {
 kSimFtmTrgSrc0,
 kSimFtmTrgSrc1 }

SIM FlexTimer x trigger y select.

- enum `sim_ftm_clk_sel_k22f25612_t` {
 kSimFtmClkSel0,
 kSimFtmClkSel1 }

SIM FlexTimer external clock select.

- enum `sim_ftm_ch_src_k22f25612_t` {
 kSimFtmChSrc0,
 kSimFtmChSrc1,
 kSimFtmChSrc2,
 kSimFtmChSrc3 }

SIM FlexTimer x channel y input capture source select.

- enum `sim_ftm_ch_out_src_k22f25612_t` {
 kSimFtmChOutSrc0,
 kSimFtmChOutSrc1 }

- *SIM FlexTimer x channel y output source select.*
• enum `sim_ftm_ftl_sel_k22f25612_t` {
`kSimFtmFltSel0,`
`kSimFtmFltSel1 }`
- *SIM FlexTimer x Fault y select.*
• enum `sim_tpm_clk_sel_k22f25612_t` {
`kSimTpmClkSel0,`
`kSimTpmClkSel1 }`
- *SIM Timer/PWM external clock select.*
• enum `sim_tpm_ch_src_k22f25612_t` {
`kSimTpmChSrc0,`
`kSimTpmChSrc1 }`
- *SIM Timer/PWM x channel y input capture source select.*
• enum `sim_cmtuartpad_streng_k22f25612_t` {
`kSimCmtuartSinglePad,`
`kSimCmtuartDualPad }`
- *SIM CMT/UART pad drive strength.*
• enum `sim_ptd7pad_streng_k22f25612_t` {
`kSimPtd7padSinglePad,`
`kSimPtd7padDualPad }`
- *SIM PTD7 pad drive strength.*
• enum `sim_flexport_security_level_k22f25612_t` {
`kSimFbslLevel0,`
`kSimFbslLevel1,`
`kSimFbslLevel2,`
`kSimFbslLevel3 }`
- *SIM FlexBus security level.*
• enum `sim_clock_gate_name_k22f25612_t`
Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

40.2.9.2 Macro Definition Documentation

40.2.9.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.9.3 Enumeration Type Documentation

40.2.9.3.1 enum clock_wdog_src_k22f25612_t

Enumerator

`kClockWdogSrcLpoClk` LPO.

`kClockWdogSrcAltClk` Alternative clock, for this SOC it is Bus clock.

SIM HAL driver

40.2.9.3.2 enum clock_trace_src_k22f25612_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.9.3.3 enum clock_port_filter_src_k22f25612_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.9.3.4 enum clock_lptmr_src_k22f25612_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

kClockLptmrSrcLpoClk LPO clock.

kClockLptmrSrcEr32kClk ERCLK32K clock.

kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

40.2.9.3.5 enum clock_usbfs_src_k22f25612_t

Enumerator

kClockUsbfsSrcExt External bypass clock (USB_CLKIN)

kClockUsbfsSrcPllFllSel Clock divider USB FS clock.

40.2.9.3.6 enum clock_lpuart_src_k22f25612_t

Enumerator

kClockLpuartSrcNone Clock disabled.

kClockLpuartSrcPllFllSel Clock as selected by SOPT2[PLLFLSEL].

kClockLpuartSrcOsc0erClk OSCERCLK.

kClockLpuartSrcMcgIrClk MCGIRCLK.

40.2.9.3.7 enum clock_sai_src_k22f25612_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllFllSel MCGPLLCLK.

40.2.9.3.8 enum clock_pllfl_sel_k22f25612_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.
kClockPllFllSelIrc48M IRC48MCLK.

40.2.9.3.9 enum clock_er32k_src_k22f25612_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

40.2.9.3.10 enum clock_clkout_src_k22f25612_t

Enumerator

kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelRtc RTC 32k clock.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

40.2.9.3.11 enum clock_rtcout_src_k22f25612_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

SIM HAL driver

40.2.9.3.12 enum clock_osc32kout_sel_k22f25612_t

Enumerator

kClockOsc32koutNone ERCLK32K is not output.

kClockOsc32koutPte0 ERCLK32K is output on PTE0.

kClockOsc32koutPte26 ERCLK32K is output on PTE26.

40.2.9.3.13 enum sim_usbsstby_mode_k22f25612_t

Enumerator

kSimUsbsstbyNoRegulator regulator not in standby during Stop modes

kSimUsbsstbyWithRegulator regulator in standby during Stop modes

40.2.9.3.14 enum sim_usbvstby_mode_k22f25612_t

Enumerator

kSimUsbvstbyNoRegulator regulator not in standby during VLPR and VLPW modes

kSimUsbvstbyWithRegulator regulator in standby during VLPR and VLPW modes

40.2.9.3.15 enum sim_adc_pretrg_sel_k22f25612_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.

kSimAdcPretrgselB Pre-trigger B selected for ADCx.

40.2.9.3.16 enum sim_adc_trg_sel_k22f25612_t

Enumerator

kSimAdcTrgselExt External trigger.

kSimAdcTrgSelHighSpeedComp0 High speed comparator 0 output.

kSimAdcTrgSelHighSpeedComp1 High speed comparator 1 output.

kSimAdcTrgSelPit0 PIT trigger 0.

kSimAdcTrgSelPit1 PIT trigger 1.

kSimAdcTrgSelPit2 PIT trigger 2.

kSimAdcTrgSelPit3 PIT trigger 3.

kSimAdcTrgSelFtm0 FTM0 trigger.

kSimAdcTrgSelFtm1 FTM1 trigger.

kSimAdcTrgSelFtm2 FTM2 trigger.

kSimAdcTrgSelRtcAlarm RTC alarm.
kSimAdcTrgSelRtcSec RTC seconds.
kSimAdcTrgSelLptimer Low-power timer trigger.

40.2.9.3.17 enum sim_lpuart_rxsrv_k22f25612_t

Enumerator

kSimLpuartRxsrvPin LPUARTx_RX Pin.
kSimLpuartRxsrvCmp0 CMP0.
kSimLpuartRxsrvCmp1 CMP1.

40.2.9.3.18 enum sim_uart_rxsrv_k22f25612_t

Enumerator

kSimUartRxsrvPin UARTx_RX Pin.
kSimUartRxsrvCmp0 CMP0.
kSimUartRxsrvCmp1 CMP1.

40.2.9.3.19 enum sim_uart_txsrc_k22f25612_t

Enumerator

kSimUartTxsrcPin UARTx_TX Pin.
kSimUartTxsrcFtm1 UARTx_TX pin modulated with FTM1 channel 0 output.
kSimUartTxsrcFtm2 UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.9.3.20 enum sim_ftm_trg_src_k22f25612_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

40.2.9.3.21 enum sim_ftm_clk_sel_k22f25612_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

SIM HAL driver

40.2.9.3.22 enum sim_ftm_ch_src_k22f25612_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

40.2.9.3.23 enum sim_ftm_ch_out_src_k22f25612_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source selection 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source selection 1.

40.2.9.3.24 enum sim_ftm_flt_sel_k22f25612_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

40.2.9.3.25 enum sim_tpm_clk_sel_k22f25612_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

40.2.9.3.26 enum sim_tpm_ch_src_k22f25612_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

40.2.9.3.27 enum sim_cmtuartpad_strenght_k22f25612_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

40.2.9.3.28 enum sim_ptd7pad_strength_k22f25612_t

Enumerator

kSimPtd7padSinglePad Single-pad drive strength for PTD7.

kSimPtd7padDualPad Dual-pad drive strength for PTD7.

40.2.9.3.29 enum sim_flexbus_security_level_k22f25612_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

40.2.9.3.30 enum sim_clock_gate_name_k22f25612_t

40.2.10 K22F51212 SIM HAL driver

40.2.10.1 Overview

The section describes the enumerations, macros and data structures for K22F51212 SIM HAL driver.

Files

- file [fsl_sim_hal_MK22F51212.h](#)

Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k22f51212_t](#) {
 kClockWdogSrcLpoClk,
 kClockWdogSrcAltClk }
WDOG clock source select.
- enum [clock_trace_src_k22f51212_t](#) {
 kClockTraceSrcMcgoutClk,
 kClockTraceSrcCoreClk }
Debug trace clock source select.
- enum [clock_port_filter_src_k22f51212_t](#) {
 kClockPortFilterSrcBusClk,
 kClockPortFilterSrcLpoClk }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k22f51212_t](#) {
 kClockLptmrSrcMcgIrClk,
 kClockLptmrSrcLpoClk,
 kClockLptmrSrcEr32kClk,
 kClockLptmrSrcOsc0erClkUndiv }
LPTMR clock source select.
- enum [clock_usbfs_src_k22f51212_t](#) {
 kClockUsbfsSrcExt,
 kClockUsbfsSrcPllFllSel }
SIM USB FS clock source.
- enum [clock_lpuart_src_k22f51212_t](#) {
 kClockLpuartSrcNone,
 kClockLpuartSrcPllFllSel,
 kClockLpuartSrcOsc0erClk,
 kClockLpuartSrcMcgIrClk }
SIM LPUART clock source.

- enum `clock_sai_src_k22f51212_t` {

 `kClockSaiSrcSysClk` = 0U,

 `kClockSaiSrcOsc0erClk` = 1U,

 `kClockSaiSrcPllFllSel` = 3U }

 SAI clock source.
- enum `clock_pllfl_sel_k22f51212_t` {

 `kClockPllFllSelFll` = 0U,

 `kClockPllFllSelPll` = 1U,

 `kClockPllFllSelIrc48M` = 3U }

 SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k22f51212_t` {

 `kClockEr32kSrcOsc0` = 0U,

 `kClockEr32kSrcRtc` = 2U,

 `kClockEr32kSrcLpo` = 3U }

 SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k22f51212_t` {

 `kClockClkoutSelFlexbusClk` = 0U,

 `kClockClkoutSelFlashClk` = 2U,

 `kClockClkoutSelLpoClk` = 3U,

 `kClockClkoutSelMcgIrClk` = 4U,

 `kClockClkoutSelRtc` = 5U,

 `kClockClkoutSelOsc0erClk` = 6U,

 `kClockClkoutSelIrc48M` = 7U }

 SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k22f51212_t` {

 `kClockRtcoutSrc1Hz`,

 `kClockRtcoutSrc32kHz` }

 SIM RTCCLKOUTSEL clock source select.
- enum `clock_osc32kout_sel_k22f51212_t` {

 `kClockOsc32koutNone` = 0U,

 `kClockOsc32koutPte0` = 1U,

 `kClockOsc32koutPte26` = 2U }

 SIM OSC32KOUT selection.
- enum `sim_usbsstby_mode_k22f51212_t` {

 `kSimUsbstbyNoRegulator`,

 `kSimUsbstbyWithRegulator` }

 SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k22f51212_t` {

 `kSimUsbvstbyNoRegulator`,

 `kSimUsbvstbyWithRegulator` }

 SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
- enum `sim_adc_pretrg_sel_k22f51212_t` {

 `kSimAdcPretrgselA`,

 `kSimAdcPretrgselB` }

 SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_k22f51212_t` {

SIM HAL driver

```
kSimAdcTrgSelExt = 0U,  
kSimAdcTrgSelHighSpeedComp0 = 1U,  
kSimAdcTrgSelHighSpeedComp1 = 2U,  
kSimAdcTrgSelPit0 = 4U,  
kSimAdcTrgSelPit1 = 5U,  
kSimAdcTrgSelPit2 = 6U,  
kSimAdcTrgSelPit3 = 7U,  
kSimAdcTrgSelFtm0 = 8U,  
kSimAdcTrgSelFtm1 = 9U,  
kSimAdcTrgSelFtm2 = 10U,  
kSimAdcTrgSelFtm3 = 11U,  
kSimAdcTrgSelRtcAlarm = 12U,  
kSimAdcTrgSelRtcSec = 13U,  
kSimAdcTrgSelLptimer = 14U }
```

SIM ADCx trigger select.

- enum `sim_lpuart_rxsrc_k22f51212_t` {
 kSimLpuartRxsrcPin,
 kSimLpuartRxsrcCmp0,
 kSimLpuartRxsrcCmp1 }

SIM LPUART RX source.

- enum `sim_uart_rxsrc_k22f51212_t` {
 kSimUartRxsrcPin,
 kSimUartRxsrcCmp0,
 kSimUartRxsrcCmp1 }

SIM receive data source select.

- enum `sim_uart_txsrc_k22f51212_t` {
 kSimUartTxsrcPin,
 kSimUartTxsrcFtm1,
 kSimUartTxsrcFtm2 }

SIM transmit data source select.

- enum `sim_ftm_trg_src_k22f51212_t` {
 kSimFtmTrgSrc0,
 kSimFtmTrgSrc1 }

SIM FlexTimer x trigger y select.

- enum `sim_ftm_clk_sel_k22f51212_t` {
 kSimFtmClkSel0,
 kSimFtmClkSel1 }

SIM FlexTimer external clock select.

- enum `sim_ftm_ch_src_k22f51212_t` {
 kSimFtmChSrc0,
 kSimFtmChSrc1,
 kSimFtmChSrc2,
 kSimFtmChSrc3 }

SIM FlexTimer x channel y input capture source select.

- enum `sim_ftm_ch_out_src_k22f51212_t` {
 kSimFtmChOutSrc0,

- ```

kSimFtmChOutSrc1 }

 SIM FlexTimer x channel y output source select.
• enum sim_ftm_ftl_sel_k22f51212_t {
 kSimFtmFltSel0,
 kSimFtmFltSel1 }

 SIM FlexTimer x Fault y select.
• enum sim_tpm_clk_sel_k22f51212_t {
 kSimTpmClkSel0,
 kSimTpmClkSel1 }

 SIM Timer/PWM external clock select.
• enum sim_tpm_ch_src_k22f51212_t {
 kSimTpmChSrc0,
 kSimTpmChSrc1,
 kSimTpmChSrc2,
 kSimTpmChSrc3 }

 SIM Timer/PWM x channel y input capture source select.
• enum sim_cmtuartpad_strenght_k22f51212_t {
 kSimCmtuartSinglePad,
 kSimCmtuartDualPad }

 SIM CMT/UART pad drive strength.
• enum sim_ptd7pad_strenght_k22f51212_t {
 kSimPtd7padSinglePad,
 kSimPtd7padDualPad }

 SIM PTD7 pad drive strength.
• enum sim_flexbus_security_level_k22f51212_t {
 kSimFbslLevel0,
 kSimFbslLevel1,
 kSimFbslLevel2,
 kSimFbslLevel3 }

 SIM FlexBus security level.
• enum sim_clock_gate_name_k22f51212_t
 Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

```

#### 40.2.10.2 Macro Definition Documentation

##### 40.2.10.2.1 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

#### 40.2.10.3 Enumeration Type Documentation

##### 40.2.10.3.1 enum clock\_wdog\_src\_k22f51212\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 40.2.10.3.2 enum clock\_trace\_src\_k22f51212\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

### 40.2.10.3.3 enum clock\_port\_filter\_src\_k22f51212\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

### 40.2.10.3.4 enum clock\_lptmr\_src\_k22f51212\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 40.2.10.3.5 enum clock\_usbfs\_src\_k22f51212\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

### 40.2.10.3.6 enum clock\_lpuart\_src\_k22f51212\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.

*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLSEL].

*kClockLpuartSrcOsc0erClk* OSCERCLK.

*kClockLpuartSrcMcgIrClk* MCGIRCLK.

**40.2.10.3.7 enum clock\_sai\_src\_k22f51212\_t**

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFllSel* MCGPLLFLCLK.

**40.2.10.3.8 enum clock\_pllfl\_sel\_k22f51212\_t**

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

**40.2.10.3.9 enum clock\_er32k\_src\_k22f51212\_t**

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

**40.2.10.3.10 enum clock\_clkout\_src\_k22f51212\_t**

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

**40.2.10.3.11 enum clock\_rtcout\_src\_k22f51212\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

## SIM HAL driver

### 40.2.10.3.12 enum clock\_osc32kout\_sel\_k22f51212\_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 40.2.10.3.13 enum sim\_usbsstby\_mode\_k22f51212\_t

Enumerator

- kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes
- kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 40.2.10.3.14 enum sim\_usbvstby\_mode\_k22f51212\_t

Enumerator

- kSimUsvbstbyNoRegulator* regulator not in standby during VLPR and VLPW modes
- kSimUsvbstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 40.2.10.3.15 enum sim\_adc\_pretrg\_sel\_k22f51212\_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 40.2.10.3.16 enum sim\_adc\_trg\_sel\_k22f51212\_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.

*kSimAdcTrgSelFtm3* FTM3 trigger.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

#### 40.2.10.3.17 enum sim\_lpuart\_rxsrv\_k22f51212\_t

Enumerator

*kSimLpuartRxsrvPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrvCmp0* CMP0.  
*kSimLpuartRxsrvCmp1* CMP1.

#### 40.2.10.3.18 enum sim\_uart\_rxsrv\_k22f51212\_t

Enumerator

*kSimUartRxsrvPin* UARTx\_RX Pin.  
*kSimUartRxsrvCmp0* CMP0.  
*kSimUartRxsrvCmp1* CMP1.

#### 40.2.10.3.19 enum sim\_uart\_txsrc\_k22f51212\_t

Enumerator

*kSimUartTxsrcPin* UARTx\_TX Pin.  
*kSimUartTxsrcFtm1* UARTx\_TX pin modulated with FTM1 channel 0 output.  
*kSimUartTxsrcFtm2* UARTx\_TX pin modulated with FTM2 channel 0 output.

#### 40.2.10.3.20 enum sim\_ftm\_trg\_src\_k22f51212\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

#### 40.2.10.3.21 enum sim\_ftm\_clk\_sel\_k22f51212\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

## SIM HAL driver

### 40.2.10.3.22 enum sim\_ftm\_ch\_src\_k22f51212\_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

### 40.2.10.3.23 enum sim\_ftm\_ch\_out\_src\_k22f51212\_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source 1.

### 40.2.10.3.24 enum sim\_ftm\_flt\_sel\_k22f51212\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 40.2.10.3.25 enum sim\_tpm\_clk\_sel\_k22f51212\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 40.2.10.3.26 enum sim\_tpm\_ch\_src\_k22f51212\_t

Enumerator

- kSimTpmChSrc0* TPM x channel y input capture source 0.
- kSimTpmChSrc1* TPM x channel y input capture source 1.
- kSimTpmChSrc2* TPM x channel y input capture source 2.
- kSimTpmChSrc3* TPM x channel y input capture source 3.

### 40.2.10.3.27 enum sim\_cmtuartpad\_strenght\_k22f51212\_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**40.2.10.3.28 enum sim\_ptd7pad\_strenght\_k22f51212\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.

*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**40.2.10.3.29 enum sim\_flexbus\_security\_level\_k22f51212\_t**

Enumerator

*kSimFbslLevel0* FlexBus security level 0.

*kSimFbslLevel1* FlexBus security level 1.

*kSimFbslLevel2* FlexBus security level 2.

*kSimFbslLevel3* FlexBus security level 3.

**40.2.10.3.30 enum sim\_clock\_gate\_name\_k22f51212\_t**

## SIM HAL driver

### 40.2.11 KL03Z4 SIM HAL driver

#### 40.2.11.1 Overview

The section describes the enumerations, macros and data structures for KL03Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL03Z4.h](#)

#### Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*

#### Enumerations

- enum `clock_cop_src_kl03z4_t` {  
  `kClockCopSrcLpoClk`,  
  `kClockCopSrcMcgIrClk`,  
  `kClockCopSrcOsc0erClk`,  
  `kClockCopSrcBusClk` }  
*COP clock source selection.*
- enum `clock_er32k_src_kl03z4_t` {  
  `kClockEr32kSrcOsc0` = 0U,  
  `kClockEr32kSrcRtc` = 2U,  
  `kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_osc32kout_sel_kl03z4_t` {  
  `kClockOsc32koutNone`,  
  `kClockOsc32koutPtb13` }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum `clock_lpuart_src_kl03z4_t` {  
  `kClockLpuartSrcNone`,  
  `kClockLpuartSrcIrc48M`,  
  `kClockLpuartSrcOsc0erClk`,  
  `kClockLpuartSrcMcgIrClk` }  
*SIM LPUART0 clock source.*
- enum `clock_tpm_src_kl03z4_t` {  
  `kClockTpmSrcNone`,  
  `kClockTpmSrcIrc48M`,  
  `kClockTpmSrcOsc0erClk`,  
  `kClockTpmSrcMcgIrClk` }  
*SIM TPM clock source.*

- enum `clock_lptmr_src_kl03z4_t` {
   
  `kClockLptmrSrcMcgIrClk,`
  
  `kClockLptmrSrcLpoClk,`
  
  `kClockLptmrSrcEr32kClk,`
  
  `kClockLptmrSrcOsc0erClk }`
  
    *LPTMR clock source select.*
- enum `clock_clkout_src_kl03z4_t` {
   
  `kClockClkoutSelFlashClk = 2U,`
  
  `kClockClkoutSelLpoClk = 3U,`
  
  `kClockClkoutSelMcgIrClk = 4U,`
  
  `kClockClkoutSelOsc0erClk = 6U,`
  
  `kClockClkoutSelIrc48M = 7U }`
  
    *SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kl03z4_t` {
   
  `kClockRtcoutSrc1Hz,`
  
  `kClockRtcoutSrcOsc0erClk }`
  
    *SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl03z4_t` {
   
  `kSimAdcPretrgselA,`
  
  `kSimAdcPretrgselB }`
  
    *SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl03z4_t` {
   
  `kSimAdcTrgselExt = 0U,`
  
  `kSimAdcTrgSelHighSpeedComp0 = 1U,`
  
  `kSimAdcTrgSelTpm0 = 8U,`
  
  `kSimAdcTrgSelTpm1 = 9U,`
  
  `kSimAdcTrgSelRtcAlarm = 12U,`
  
  `kSimAdcTrgSelRtcSec = 13U,`
  
  `kSimAdcTrgSelLptimer = 14U }`
  
    *SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_kl03z4_t` {
   
  `kSimUartRxsrcPin,`
  
  `kSimUartRxsrcCmp0,`
  
  `kSimUartRxsrcCmp1 }`
  
    *SIM receive data source select.*
- enum `sim_uart_txsrc_kl03z4_t` {
   
  `kSimUartTxsrcPin,`
  
  `kSimUartTxsrcFtm1,`
  
  `kSimUartTxsrcFtm2 }`
  
    *SIM transmit data source select.*
- enum `sim_tpm_clk_sel_kl03z4_t` {
   
  `kSimTpmClkSel0,`
  
  `kSimTpmClkSel1 }`
  
    *SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl03z4_t` {
   
  `kSimTpmChSrc0,`
  
  `kSimTpmChSrc1 }`

## SIM HAL driver

- *SIM Timer/PWM x channel y input capture source select.*
  - enum `sim_ptd7pad_strength_kl03z4_t` {  
    `kSimPtd7padSinglePad`,  
    `kSimPtd7padDualPad` }
- *SIM PTD7 pad drive strength.*
  - enum `sim_clock_gate_name_kl03z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 40.2.11.2 Macro Definition Documentation

#### 40.2.11.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 40.2.11.3 Enumeration Type Documentation

#### 40.2.11.3.1 enum `clock_cop_src_kl03z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO clock,1K HZ.  
*kClockCopSrcMcgIrcClk* MCG IRC Clock.  
*kClockCopSrcOsc0erClk* OSCER Clock.  
*kClockCopSrcBusClk* BUS clock.

#### 40.2.11.3.2 enum `clock_er32k_src_kl03z4_t`

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

#### 40.2.11.3.3 enum `clock_osc32kout_sel_kl03z4_t`

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPtb13* ERCLK32K output on PTB13.

#### 40.2.11.3.4 enum `clock_lpuart_src_kl03z4_t`

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.

*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLp uartSrcMcgIrClk* MCGIR clock.

#### 40.2.11.3.5 enum clock\_tpm\_src\_kl03z4\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

#### 40.2.11.3.6 enum clock\_lptmr\_src\_kl03z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 40.2.11.3.7 enum clock\_clkout\_src\_kl03z4\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

#### 40.2.11.3.8 enum clock\_rtcout\_src\_kl03z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

#### 40.2.11.3.9 enum sim\_adc\_pretrg\_sel\_kl03z4\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

## SIM HAL driver

### 40.2.11.3.10 enum sim\_adc\_trg\_sel\_kl03z4\_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelTpm0* TPM0 trigger.
- kSimAdcTrgSelTpm1* TPM1 trigger.
- kSimAdcTrgSelRtcAlarm* RTC alarm.
- kSimAdcTrgSelRtcSec* RTC seconds.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

### 40.2.11.3.11 enum sim\_uart\_rxsrv\_kl03z4\_t

Enumerator

- kSimUartRxsrvPin* UARTx\_RX Pin.
- kSimUartRxsrvCmp0* CMP0.
- kSimUartRxsrvCmp1* CMP1.

### 40.2.11.3.12 enum sim\_uart\_txsrc\_kl03z4\_t

Enumerator

- kSimUartTxsrcPin* UARTx\_TX Pin.
- kSimUartTxsrcFtm1* UARTx\_TX pin modulated with FTM1 channel 0 output.
- kSimUartTxsrcFtm2* UARTx\_TX pin modulated with FTM2 channel 0 output.

### 40.2.11.3.13 enum sim\_tpm\_clk\_sel\_kl03z4\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 40.2.11.3.14 enum sim\_tpm\_ch\_src\_kl03z4\_t

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**40.2.11.3.15 enum sim\_ptd7pad\_strenght\_kl03z4\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.

*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**40.2.11.3.16 enum sim\_clock\_gate\_name\_kl03z4\_t**

## SIM HAL driver

### 40.2.12 K24F12 SIM HAL driver

#### 40.2.12.1 Overview

The section describes the enumerations, macros and data structures for K24F12 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK24F12.h](#)

#### Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k24f12\\_t](#) {  
  kClockWdogSrcLpoClk,  
  kClockWdogSrcAltClk }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k24f12\\_t](#) {  
  kClockTraceSrcMcgoutClk,  
  kClockTraceSrcCoreClk }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k24f12\\_t](#) {  
  kClockPortFilterSrcBusClk,  
  kClockPortFilterSrcLpoClk }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k24f12\\_t](#) {  
  kClockLptmrSrcMcgIrClk,  
  kClockLptmrSrcLpoClk,  
  kClockLptmrSrcEr32kClk,  
  kClockLptmrSrcOsc0erClk }  
*LPTMR clock source select.*
- enum [clock\\_usbfs\\_src\\_k24f12\\_t](#) {  
  kClockUsbfsSrcExt,  
  kClockUsbfsSrcPllFllSel }  
*SIM USB FS clock source.*
- enum [clock\\_flexcan\\_src\\_k24f12\\_t](#) {  
  kClockFlexcanSrcOsc0erClk,  
  kClockFlexcanSrcBusClk }  
*FLEXCAN clock source select.*
- enum [clock\\_sdhc\\_src\\_k24f12\\_t](#) {

- ```

kClockSdhcSrcCoreSysClk,
kClockSdhcSrcPllFllSel,
kClockSdhcSrcOsc0erClk,
kClockSdhcSrcExt }

SDHC clock source.
• enum clock_src_k24f12_t {
  kClockSaiSrcSysClk = 0U,
  kClockSaiSrcOsc0erClk = 1U,
  kClockSaiSrcPllClk = 3U }

SAI clock source.
• enum clock_pllfl_sel_k24f12_t {
  kClockPllFllSelFll = 0U,
  kClockPllFllSelPll = 1U,
  kClockPllFllSelIrc48M = 3U }

SIM PLLFLLSEL clock source select.
• enum clock_er32k_src_k24f12_t {
  kClockEr32kSrcOsc0 = 0U,
  kClockEr32kSrcRtc = 2U,
  kClockEr32kSrcLpo = 3U }

SIM external reference clock source select (OSC32KSEL).
• enum clock_clkout_src_k24f12_t {
  kClockClkoutSelFlexbusClk = 0U,
  kClockClkoutSelFlashClk = 2U,
  kClockClkoutSelLpoClk = 3U,
  kClockClkoutSelMcgIrClk = 4U,
  kClockClkoutSelRtc = 5U,
  kClockClkoutSelOsc0erClk = 6U,
  kClockClkoutSelIrc48M = 7U }

SIM CLKOUT_SEL clock source select.
• enum clock_rtcout_src_k24f12_t {
  kClockRtcoutSrc1Hz,
  kClockRtcoutSrc32kHz }

SIM RTCCLKOUTSEL clock source select.
• enum sim_usbsstby_mode_k24f12_t {
  kSimUsbstbyNoRegulator,
  kSimUsbstbyWithRegulator }

SIM USB voltage regulator in standby mode setting during stop modes.
• enum sim_usbvstby_mode_k24f12_t {
  kSimUsbstbyNoRegulator,
  kSimUsbstbyWithRegulator }

SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
• enum sim_adc_pretrg_sel_k24f12_t {
  kSimAdcPretrgselA,
  kSimAdcPretrgselB }

SIM ADCx pre-trigger select.
• enum sim_adc_trg_sel_k24f12_t {

```

SIM HAL driver

```
kSimAdcTrgSelExt = 0U,  
kSimAdcTrgSelHighSpeedComp0 = 1U,  
kSimAdcTrgSelHighSpeedComp1 = 2U,  
kSimAdcTrgSelHighSpeedComp2 = 3U,  
kSimAdcTrgSelPit0 = 4U,  
kSimAdcTrgSelPit1 = 5U,  
kSimAdcTrgSelPit2 = 6U,  
kSimAdcTrgSelPit3 = 7U,  
kSimAdcTrgSelFtm0 = 8U,  
kSimAdcTrgSelFtm1 = 9U,  
kSimAdcTrgSelFtm2 = 10U,  
kSimAdcTrgSelFtm3 = 11U,  
kSimAdcTrgSelRtcAlarm = 12U,  
kSimAdcTrgSelRtcSec = 13U,  
kSimAdcTrgSelLptimer = 14U }
```

SIM ADCx trigger select.

- enum `sim_uart_rxsrc_k24f12_t`{
 `kSimUartRxsrcPin,`
 `kSimUartRxsrcCmp0,`
 `kSimUartRxsrcCmp1` }

SIM receive data source select.

- enum `sim_uart_txsrc_k24f12_t`{
 `kSimUartTxsrcPin,`
 `kSimUartTxsrcFtm1,`
 `kSimUartTxsrcFtm2` }

SIM transmit data source select.

- enum `sim_ftm_trg_src_k24f12_t`{
 `kSimFtmTrgSrc0,`
 `kSimFtmTrgSrc1` }

SIM FlexTimer x trigger y select.

- enum `sim_ftm_clk_sel_k24f12_t`{
 `kSimFtmClkSel0,`
 `kSimFtmClkSel1` }

SIM FlexTimer external clock select.

- enum `sim_ftm_ch_src_k24f12_t`{
 `kSimFtmChSrc0,`
 `kSimFtmChSrc1,`
 `kSimFtmChSrc2,`
 `kSimFtmChSrc3` }

SIM FlexTimer x channel y input capture source select.

- enum `sim_ftm_flt_sel_k24f12_t`{
 `kSimFtmFiltSel0,`
 `kSimFtmFiltSel1` }

SIM FlexTimer x Fault y select.

- enum `sim_tpm_clk_sel_k24f12_t`{
 `kSimTpmClkSel0,`

```

kSimTpmClkSel1 }

    SIM Timer/PWM external clock select.
• enum sim_tpm_ch_src_k24f12_t {
    kSimTpmChSrc0,
    kSimTpmChSrc1,
    kSimTpmChSrc2,
    kSimTpmChSrc3 }

    SIM Timer/PWM x channel y input capture source select.
• enum sim_cmtuartpad_strenght_k24f12_t {
    kSimCmtuartSinglePad,
    kSimCmtuartDualPad }

    SIM CMT/UART pad drive strength.
• enum sim_ptd7pad_strenght_k24f12_t {
    kSimPtd7padSinglePad,
    kSimPtd7padDualPad }

    SIM PTD7 pad drive strength.
• enum sim_flexbus_security_level_k24f12_t {
    kSimFbslLevel0,
    kSimFbslLevel1,
    kSimFbslLevel2,
    kSimFbslLevel3 }

    SIM FlexBus security level.
• enum sim_clock_gate_name_k24f12_t
    Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

```

40.2.12.2 Macro Definition Documentation

40.2.12.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.12.3 Enumeration Type Documentation

40.2.12.3.1 enum clock_wdog_src_k24f12_t

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

40.2.12.3.2 enum clock_trace_src_k24f12_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.12.3.3 enum clock_port_filter_src_k24f12_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.12.3.4 enum clock_lptmr_src_k24f12_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

kClockLptmrSrcLpoClk LPO clock.

kClockLptmrSrcEr32kClk ERCLK32K clock.

kClockLptmrSrcOsc0erClk OSCERCLK clock.

40.2.12.3.5 enum clock_usbfs_src_k24f12_t

Enumerator

kClockUsbfsSrcExt External bypass clock (USB_CLKIN)

kClockUsbfsSrcPllFllSel Clock divider USB FS clock.

40.2.12.3.6 enum clock_flexcan_src_k24f12_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.

kClockFlexcanSrcBusClk Bus clock.

40.2.12.3.7 enum clock_sdhc_src_k24f12_t

Enumerator

kClockSdhcSrcCoreSysClk Core/system clock.

kClockSdhcSrcPllFllSel clock as selected by SOPT2[PLLFLLSEL].

kClockSdhcSrcOsc0erClk OSCERCLK clock.

kClockSdhcSrcExt External bypass clock (SDHC0_CLKIN)

40.2.12.3.8 enum clock_sai_src_k24f12_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

40.2.12.3.9 enum clock_pllfll_sel_k24f12_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.
kClockPllFllSelIrc48M IRC48MCLK.

40.2.12.3.10 enum clock_er32k_src_k24f12_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

40.2.12.3.11 enum clock_clkout_src_k24f12_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelRtc RTC 32k clock.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

40.2.12.3.12 enum clock_rtcout_src_k24f12_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

SIM HAL driver

40.2.12.3.13 enum sim_usbsstby_mode_k24f12_t

Enumerator

kSimUsbsstbyNoRegulator regulator not in standby during Stop modes

kSimUsbsstbyWithRegulator regulator in standby during Stop modes

40.2.12.3.14 enum sim_usbvstby_mode_k24f12_t

Enumerator

kSimUsbvstbyNoRegulator regulator not in standby during VLPR and VLPW modes

kSimUsbvstbyWithRegulator regulator in standby during VLPR and VLPW modes

40.2.12.3.15 enum sim_adc_pretrg_sel_k24f12_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.

kSimAdcPretrgselB Pre-trigger B selected for ADCx.

40.2.12.3.16 enum sim_adc_trg_sel_k24f12_t

Enumerator

kSimAdcTrgselExt External trigger.

kSimAdcTrgSelHighSpeedComp0 High speed comparator 0 output.

kSimAdcTrgSelHighSpeedComp1 High speed comparator 1 output.

kSimAdcTrgSelHighSpeedComp2 High speed comparator 2 output.

kSimAdcTrgSelPit0 PIT trigger 0.

kSimAdcTrgSelPit1 PIT trigger 1.

kSimAdcTrgSelPit2 PIT trigger 2.

kSimAdcTrgSelPit3 PIT trigger 3.

kSimAdcTrgSelFtm0 FTM0 trigger.

kSimAdcTrgSelFtm1 FTM1 trigger.

kSimAdcTrgSelFtm2 FTM2 trigger.

kSimAdcTrgSelFtm3 FTM3 trigger.

kSimAdcTrgSelRtcAlarm RTC alarm.

kSimAdcTrgSelRtcSec RTC seconds.

kSimAdcTrgSelLptimer Low-power timer trigger.

40.2.12.3.17 enum sim_uart_rxsrv_k24f12_t

Enumerator

- kSimUartRxsrvPin* UARTx_RX Pin.
- kSimUartRxsrvCmp0* CMP0.
- kSimUartRxsrvCmp1* CMP1.

40.2.12.3.18 enum sim_uart_txsrc_k24f12_t

Enumerator

- kSimUartTxsrcPin* UARTx_TX Pin.
- kSimUartTxsrcFtm1* UARTx_TX pin modulated with FTM1 channel 0 output.
- kSimUartTxsrcFtm2* UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.12.3.19 enum sim_ftm_trg_src_k24f12_t

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

40.2.12.3.20 enum sim_ftm_clk_sel_k24f12_t

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

40.2.12.3.21 enum sim_ftm_ch_src_k24f12_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

40.2.12.3.22 enum sim_ftm_flt_sel_k24f12_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

SIM HAL driver

40.2.12.3.23 enum sim_tpm_clk_sel_k24f12_t

Enumerator

kSimTpmClkSel0 Timer/PWM TPM_CLKIN0 pin.

kSimTpmClkSel1 Timer/PWM TPM_CLKIN1 pin.

40.2.12.3.24 enum sim_tpm_ch_src_k24f12_t

Enumerator

kSimTpmChSrc0 TPM x channel y input capture source 0.

kSimTpmChSrc1 TPM x channel y input capture source 1.

kSimTpmChSrc2 TPM x channel y input capture source 2.

kSimTpmChSrc3 TPM x channel y input capture source 3.

40.2.12.3.25 enum sim_cmtuartpad_strenght_k24f12_t

Enumerator

kSimCmtuartSinglePad Single-pad drive strength for CMT IRO or UART0_TXD.

kSimCmtuartDualPad Dual-pad drive strength for CMT IRO or UART0_TXD.

40.2.12.3.26 enum sim_ptd7pad_strenght_k24f12_t

Enumerator

kSimPtd7padSinglePad Single-pad drive strength for PTD7.

kSimPtd7padDualPad Dual-pad drive strength for PTD7.

40.2.12.3.27 enum sim_flexbus_security_level_k24f12_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

40.2.12.3.28 enum sim_clock_gate_name_k24f12_t

40.2.13 KL46Z4 SIM HAL driver

40.2.13.1 Overview

The section describes the enumerations, macros and data structures for KL46Z4 SIM HAL driver.

Files

- file [fsl_sim_hal_MKL46Z4.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_cop_src_kl46z4_t](#) {

kClockCopSrcLpoClk,

kClockCopSrcAltClk }

COP clock source select.
- enum [clock_tpm_src_kl46z4_t](#) {

kClockTpmSrcNone,

kClockTpmSrcPllFllSel,

kClockTpmSrcOsc0erClk,

kClockTpmSrcMcgIrClk }

TPM clock source select.
- enum [clock_lptmr_src_kl46z4_t](#) {

kClockLptmrSrcMcgIrClk,

kClockLptmrSrcLpoClk,

kClockLptmrSrcEr32kClk,

kClockLptmrSrcOsc0erClk }

LPTMR clock source select.
- enum [clock_lpisci_src_kl46z4_t](#) {

kClockLpisciSrcNone,

kClockLpisciSrcPllFllSel,

kClockLpisciSrcOsc0erClk,

kClockLpisciSrcMcgIrClk }

UART0 clock source select.
- enum [clock_usbfs_src_kl46z4_t](#) {

kClockUsbfsSrcExt,

kClockUsbfsSrcPllFllSel }

USB clock source select.
- enum [clock_sai_src_kl46z4_t](#) {

SIM HAL driver

- ```
kClockSaiSrcSysClk = 0U,
kClockSaiSrcOsc0erClk = 1U,
kClockSaiSrcPllClk = 3U }
 SAI clock source.
• enum clock_pllfl_sel_kl46z4_t {
 kClockPllFlSelFlI,
 kClockPllFlSelPll }
 SIM PLLFLLSEL clock source select.
• enum clock_er32k_src_kl46z4_t {
 kClockEr32kSrcOsc0 = 0U,
 kClockEr32kSrcReserved = 1U,
 kClockEr32kSrcRtc = 2U,
 kClockEr32kSrcLpo = 3U }
 SIM external reference clock source select (OSC32KSEL)
• enum clock_clkout_src_kl46z4_t {
 kClockClkoutReserved = 0U,
 kClockClkoutReserved1 = 1U,
 kClockClkoutBusClk = 2U,
 kClockClkoutLpoClk = 3U,
 kClockClkoutMcgIrClk = 4U,
 kClockClkoutReserved2 = 5U,
 kClockClkoutOsc0erClk = 6U,
 kClockClkoutReserved3 = 7U }
 SIM CLKOUT_SEL clock source select.
• enum clock_rtcout_src_kl46z4_t {
 kClockRtcoutSrc1Hz,
 kClockRtcoutSrc32kHz }
 SIM RTCCLKOUTSEL clock source select.
• enum sim_usbsstby_mode_kl46z4_t {
 kSimUsbstbyNoRegulator,
 kSimUsbstbyWithRegulator }
 SIM USB voltage regulator in standby mode setting during stop modes.
• enum sim_usbvstby_mode_kl46z4_t {
 kSimUsbvstbyNoRegulator,
 kSimUsbvstbyWithRegulator }
 SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
• enum sim_adc_pretrg_sel_kl46z4_t {
 kSimAdcPretrgselA,
 kSimAdcPretrgselB }
 SIM ADCx pre-trigger select.
• enum sim_adc_trg_sel_kl46z4_t {
```

```

kSimAdcTrgSelExt = 0U,
kSimAdcTrgSelComp0 = 1U,
kSimAdcTrgSelReserved = 2U,
kSimAdcTrgSelReserved1 = 3U,
kSimAdcTrgSelPit0 = 4U,
kSimAdcTrgSelPit1 = 5U,
kSimAdcTrgSelReserved2 = 6U,
kSimAdcTrgSelReserved3 = 7U,
kSimAdcTrgSelTpm0 = 8U,
kSimAdcTrgSelTpm1 = 9U,
kSimAdcTrgSelTpm2 = 10U,
kSimAdcTrgSelReserved4 = 11U,
kSimAdcTrgSelRtcAlarm = 12U,
kSimAdcTrgSelRtcSec = 13U,
kSimAdcTrgSelLptimer = 14U,
kSimAdcTrgSelReserved5 = 15U }

```

*SIM ADCx trigger select.*

- enum `sim_uart_rxsr_kl46z4_t` {
   
kSimUartRxsrPin,
   
kSimUartRxsrCmp0 }

*SIM receive data source select.*

- enum `sim_uart_txsr_kl46z4_t` {
   
kSimUartTxsrPin,
   
kSimUartTxsrTpm1,
   
kSimUartTxsrTpm2,
   
kSimUartTxsrReserved }

*SIM transmit data source select.*

- enum `sim_tpm_clk_sel_kl46z4_t` {
   
kSimTpmClkSel0,
   
kSimTpmClkSel1 }

*SIM Timer/PWM external clock select.*

- enum `sim_tpm_ch_src_kl46z4_t` {
   
kSimTpmChSrc0,
   
kSimTpmChSrc1,
   
kSimTpmChSrc2,
   
kSimTpmChSrc3 }

*SIM Timer/PWM x channel y input capture source select.*

- enum `sim_clock_gate_name_kl46z4_t`

*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 40.2.13.2 Macro Definition Documentation

40.2.13.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 40.2.13.3 Enumeration Type Documentation

#### 40.2.13.3.1 enum `clock_cop_src_kl46z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KL46Z4 it is Bus clock.

#### 40.2.13.3.2 enum `clock_tpm_src_kl46z4_t`

Enumerator

*kClockTpmSrcNone* clock disabled

*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].

*kClockTpmSrcOsc0erClk* OSCERCLK clock.

*kClockTpmSrcMcgIrClk* MCGIR clock.

#### 40.2.13.3.3 enum `clock_lptmr_src_kl46z4_t`

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 40.2.13.3.4 enum `clock_lpisci_src_kl46z4_t`

Enumerator

*kClockLpisciSrcNone* clock disabled

*kClockLpisciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].

*kClockLpisciSrcOsc0erClk* OSCERCLK clock.

*kClockLpisciSrcMcgIrClk* MCGIR clock.

**40.2.13.3.5 enum clock\_usvfs\_src\_kl46z4\_t**

Enumerator

*kClockUsvfsSrcExt* USB CLKIN Clock.*kClockUsvfsSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL]**40.2.13.3.6 enum clock\_sai\_src\_kl46z4\_t**

Enumerator

*kClockSaiSrcSysClk* SYSCLK.*kClockSaiSrcOsc0erClk* OSC0ERCLK.*kClockSaiSrcPllClk* MCGPLLCLK.**40.2.13.3.7 enum clock\_pllfl\_sel\_kl46z4\_t**

Enumerator

*kClockPllFllSelFll* Fll clock.*kClockPllFllSelPll* Pll0 clock.**40.2.13.3.8 enum clock\_er32k\_src\_kl46z4\_t**

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.*kClockEr32kSrcReserved* Reserved.*kClockEr32kSrcRtc* RTC 32k clock.*kClockEr32kSrcLpo* LPO clock.**40.2.13.3.9 enum clock\_clkout\_src\_kl46z4\_t**

Enumerator

*kClockClkoutReserved* Reserved.*kClockClkoutReserved1* Reserved.*kClockClkoutBusClk* Bus clock.*kClockClkoutLpoClk* LPO clock.*kClockClkoutMcgIrClk* MCG ir clock.*kClockClkoutReserved2* Reserved.*kClockClkoutOsc0erClk* OSC0ER clock.*kClockClkoutReserved3* Reserved.

## SIM HAL driver

### 40.2.13.3.10 enum clock\_rtcout\_src\_kl46z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock

*kClockRtcoutSrc32kHz* 32KHz clock

### 40.2.13.3.11 enum sim\_usbsstby\_mode\_kl46z4\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes

*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 40.2.13.3.12 enum sim\_usbvstby\_mode\_kl46z4\_t

Enumerator

*kSimUsvbstbyNoRegulator* regulator not in standby during VLPR and VLPW modes

*kSimUsvbstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 40.2.13.3.13 enum sim\_adc\_pretrg\_sel\_kl46z4\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.

*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 40.2.13.3.14 enum sim\_adc\_trg\_sel\_kl46z4\_t

Enumerator

*kSimAdcTrgselExt* External trigger.

*kSimAdcTrgSelComp0* CMP0 output.

*kSimAdcTrgSelReserved* Reserved.

*kSimAdcTrgSelReserved1* Reserved.

*kSimAdcTrgSelPit0* PIT trigger 0.

*kSimAdcTrgSelPit1* PIT trigger 1.

*kSimAdcTrgSelReserved2* Reserved.

*kSimAdcTrgSelReserved3* Reserved.

*kSimAdcTrgSelTpm0* TPM0 overflow.

*kSimAdcTrgSelTpm1* TPM1 overflow.

*kSimAdcTrgSelTpm2* TPM2 overflow.

*kSimAdcTrgSelReserved4* Reserved.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.  
*kSimAdcTrgSelReserved5* Reserved.

#### 40.2.13.3.15 enum sim\_uart\_rxsrv\_kl46z4\_t

Enumerator

*kSimUartRxsrvPin* UARTx\_RX Pin.  
*kSimUartRxsrvCmp0* CMP0.

#### 40.2.13.3.16 enum sim\_uart\_txsrc\_kl46z4\_t

Enumerator

*kSimUartTxsrcPin* UARTx\_TX Pin.  
*kSimUartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.  
*kSimUartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.  
*kSimUartTxsrcReserved* Reserved.

#### 40.2.13.3.17 enum sim\_tpm\_clk\_sel\_kl46z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

#### 40.2.13.3.18 enum sim\_tpm\_ch\_src\_kl46z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.  
*kSimTpmChSrc1* CMP0 output.  
*kSimTpmChSrc2* Reserved.  
*kSimTpmChSrc3* USB start of frame pulse.

#### 40.2.13.3.19 enum sim\_clock\_gate\_name\_kl46z4\_t

## SIM HAL driver

### 40.2.14 K24F25612 SIM HAL driver

#### 40.2.14.1 Overview

The section describes the enumerations, macros and data structures for K24F25612 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK24F25612.h](#)

#### Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k24f25612\\_t](#) {  
  kClockWdogSrcLpoClk,  
  kClockWdogSrcAltClk }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k24f25612\\_t](#) {  
  kClockTraceSrcMcgoutClk,  
  kClockTraceSrcCoreClk }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k24f25612\\_t](#) {  
  kClockPortFilterSrcBusClk,  
  kClockPortFilterSrcLpoClk }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k24f25612\\_t](#) {  
  kClockLptmrSrcMcgIrClk,  
  kClockLptmrSrcLpoClk,  
  kClockLptmrSrcEr32kClk,  
  kClockLptmrSrcOsc0erClk }  
*LPTMR clock source select.*
- enum [clock\\_usbfs\\_src\\_k24f25612\\_t](#) {  
  kClockUsbfsSrcExt,  
  kClockUsbfsSrcPliflSel }  
*SIM USB FS clock source.*
- enum [clock\\_flexcan\\_src\\_k24f25612\\_t](#) {  
  kClockFlexcanSrcOsc0erClk,  
  kClockFlexcanSrcBusClk }  
*FLEXCAN clock source select.*
- enum [clock\\_sai\\_src\\_k24f25612\\_t](#) {  
  kClockSaiSrcSysClk = 0U,  
  kClockSaiSrcOsc0erClk = 1U,

- ```

kClockSaiSrcPllClk = 3U }

SAI clock source.
• enum clock\_pllfl\_sel\_k24f25612\_t {
  kClockPllFlSelFl = 0U,
  kClockPllFlSelPll = 1U,
  kClockPllFlSelIrc48M = 3U }

  SIM PLLFLLSEL clock source select.
• enum clock\_er32k\_src\_k24f25612\_t {
  kClockEr32kSrcOsc0 = 0U,
  kClockEr32kSrcRtc = 2U,
  kClockEr32kSrcLpo = 3U }

  SIM external reference clock source select (OSC32KSEL).
• enum clock\_clkout\_src\_k24f25612\_t {
  kClockClkoutSelFlashClk = 2U,
  kClockClkoutSelLpoClk = 3U,
  kClockClkoutSelMcgIrClk = 4U,
  kClockClkoutSelRtc = 5U,
  kClockClkoutSelOsc0erClk = 6U,
  kClockClkoutSelIrc48M = 7U }

  SIM CLKOUT_SEL clock source select.
• enum clock\_rtcout\_src\_k24f25612\_t {
  kClockRtcoutSrc1Hz,
  kClockRtcoutSrc32kHz }

  SIM RTCCLKOUTSEL clock source select.
• enum sim\_usbsstby\_mode\_k24f25612\_t {
  kSimUsbstbyNoRegulator,
  kSimUsbstbyWithRegulator }

  SIM USB voltage regulator in standby mode setting during stop modes.
• enum sim\_usbvstby\_mode\_k24f25612\_t {
  kSimUsbvstbyNoRegulator,
  kSimUsbvstbyWithRegulator }

  SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
• enum sim\_adc\_pretrg\_sel\_k24f25612\_t {
  kSimAdcPretrgselA,
  kSimAdcPretrgselB }

  SIM ADCx pre-trigger select.
• enum sim\_adc\_trg\_sel\_k24f25612\_t {

```

SIM HAL driver

```
kSimAdcTrgSelExt = 0U,  
kSimAdcTrgSelHighSpeedComp0 = 1U,  
kSimAdcTrgSelHighSpeedComp1 = 2U,  
kSimAdcTrgSelPit0 = 4U,  
kSimAdcTrgSelPit1 = 5U,  
kSimAdcTrgSelPit2 = 6U,  
kSimAdcTrgSelPit3 = 7U,  
kSimAdcTrgSelFtm0 = 8U,  
kSimAdcTrgSelFtm1 = 9U,  
kSimAdcTrgSelFtm2 = 10U,  
kSimAdcTrgSelFtm3 = 11U,  
kSimAdcTrgSelRtcAlarm = 12U,  
kSimAdcTrgSelRtcSec = 13U,  
kSimAdcTrgSelLptimer = 14U }
```

SIM ADCx trigger select.

- enum `sim_uart_rxsrc_k24f25612_t` {
 kSimUartRxsrcPin,
 kSimUartRxsrcCmp0,
 kSimUartRxsrcCmp1 }

SIM receive data source select.

- enum `sim_uart_txsrc_k24f25612_t` {
 kSimUartTxsrcPin,
 kSimUartTxsrcFtm1,
 kSimUartTxsrcFtm2 }

SIM transmit data source select.

- enum `sim_ftm_trg_src_k24f25612_t` {
 kSimFtmTrgSrc0,
 kSimFtmTrgSrc1 }

SIM FlexTimer x trigger y select.

- enum `sim_ftm_clk_sel_k24f25612_t` {
 kSimFtmClkSel0,
 kSimFtmClkSel1 }

SIM FlexTimer external clock select.

- enum `sim_ftm_ch_src_k24f25612_t` {
 kSimFtmChSrc0,
 kSimFtmChSrc1,
 kSimFtmChSrc2,
 kSimFtmChSrc3 }

SIM FlexTimer x channel y input capture source select.

- enum `sim_ftm_flt_sel_k24f25612_t` {
 kSimFtmFltSel0,
 kSimFtmFltSel1 }

SIM FlexTimer x Fault y select.

- enum `sim_tpm_clk_sel_k24f25612_t` {
 kSimTpmClkSel0,
 kSimTpmClkSel1 }

- *SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k24f25612_t` {

 `kSimTpmChSrc0,`

 `kSimTpmChSrc1 }`
- *SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strenght_k24f25612_t` {

 `kSimCmtuartSinglePad,`

 `kSimCmtuartDualPad }`
- *SIM CMT/UART pad drive strength.*
- enum `sim_clock_gate_name_k24f25612_t`

 Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

40.2.14.2 Macro Definition Documentation

40.2.14.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

40.2.14.3 Enumeration Type Documentation

40.2.14.3.1 enum `clock_wdog_src_k24f25612_t`

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

40.2.14.3.2 enum `clock_trace_src_k24f25612_t`

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.14.3.3 enum `clock_port_filter_src_k24f25612_t`

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.14.3.4 enum `clock_lptmr_src_k24f25612_t`

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

SIM HAL driver

kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

40.2.14.3.5 enum `clock_usbfs_src_k24f25612_t`

Enumerator

kClockUsbfsSrcExt External bypass clock (USB_CLKIN)
kClockUsbfsSrcPllFllSel Clock divider USB FS clock.

40.2.14.3.6 enum `clock_flexcan_src_k24f25612_t`

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

40.2.14.3.7 enum `clock_sai_src_k24f25612_t`

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

40.2.14.3.8 enum `clock_pllfil_sel_k24f25612_t`

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.
kClockPllFllSelIrc48M IRC48MCLK.

40.2.14.3.9 enum `clock_er32k_src_k24f25612_t`

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

40.2.14.3.10 enum clock_clkout_src_k24f25612_t

Enumerator

kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelRtc RTC 32k clock.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

40.2.14.3.11 enum clock_rtcout_src_k24f25612_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

40.2.14.3.12 enum sim_usbsstby_mode_k24f25612_t

Enumerator

kSimUsbsstbyNoRegulator regulator not in standby during Stop modes
kSimUsbsstbyWithRegulator regulator in standby during Stop modes

40.2.14.3.13 enum sim_usbvstby_mode_k24f25612_t

Enumerator

kSimUsvbstbyNoRegulator regulator not in standby during VLPR and VLPW modes
kSimUsvbstbyWithRegulator regulator in standby during VLPR and VLPW modes

40.2.14.3.14 enum sim_adc_pretrg_sel_k24f25612_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.
kSimAdcPretrgselB Pre-trigger B selected for ADCx.

SIM HAL driver

40.2.14.3.15 enum sim_adc_trg_sel_k24f25612_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.
- kSimAdcTrgSelFtm3* FTM3 trigger.
- kSimAdcTrgSelRtcAlarm* RTC alarm.
- kSimAdcTrgSelRtcSec* RTC seconds.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

40.2.14.3.16 enum sim_uart_rxsrc_k24f25612_t

Enumerator

- kSimUartRxsrcPin* UARTx_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

40.2.14.3.17 enum sim_uart_txsrc_k24f25612_t

Enumerator

- kSimUartTxsrcPin* UARTx_TX Pin.
- kSimUartTxsrcFtm1* UARTx_TX pin modulated with FTM1 channel 0 output.
- kSimUartTxsrcFtm2* UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.14.3.18 enum sim_ftm_trg_src_k24f25612_t

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

40.2.14.3.19 enum sim_ftm_clk_sel_k24f25612_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.*kSimFtmClkSel1* FTM CLKIN1 pin.**40.2.14.3.20 enum sim_ftm_ch_src_k24f25612_t**

Enumerator

kSimFtmChSrc0 FlexTimer x channel y input capture source 0.*kSimFtmChSrc1* FlexTimer x channel y input capture source 1.*kSimFtmChSrc2* FlexTimer x channel y input capture source 2.*kSimFtmChSrc3* FlexTimer x channel y input capture source 3.**40.2.14.3.21 enum sim_ftm_flt_sel_k24f25612_t**

Enumerator

kSimFtmFltSel0 FlexTimer x fault y select 0.*kSimFtmFltSel1* FlexTimer x fault y select 1.**40.2.14.3.22 enum sim_tpm_clk_sel_k24f25612_t**

Enumerator

kSimTpmClkSel0 Timer/PWM TPM_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.**40.2.14.3.23 enum sim_tpm_ch_src_k24f25612_t**

Enumerator

kSimTpmChSrc0 TPMx_CH0 signal.*kSimTpmChSrc1* CMP0 output.**40.2.14.3.24 enum sim_cmtuartpad_strenght_k24f25612_t**

Enumerator

kSimCmtuartSinglePad Single-pad drive strength for CMT IRO or UART0_TXD.*kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

40.2.14.3.25 enum sim_clock_gate_name_k24f25612_t

40.2.15 KV10Z7 SIM HAL driver

40.2.15.1 Overview

The section describes the enumerations, macros and data structures for KV10Z7 SIM HAL driver.

Files

- file `fsl_sim_hal_MKV10Z7.h`

Macros

- #define `FSL_SIM_SCGC_BIT(SCGCx, n)` (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum `clock_wdog_src_kv10z7_t` {

`kClockWdogSrcLpoClk,`

`kClockWdogSrcAltClk }`

WDOG clock source select.
- enum `clock_lptmr_src_kv10z7_t` {

`kClockLptmrSrcMcgIrClk,`

`kClockLptmrSrcLpoClk,`

`kClockLptmrSrcEr32kClk,`

`kClockLptmrSrcOsc0erClk }`

LPTMR clock source select.
- enum `clock_er32k_src_kv10z7_t` {

`kClockEr32kSrcOsc0 = 0U,`

`kClockEr32kSrcLpo = 3U }`

SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_kv10z7_t` {

`kClockClkoutSelBusClk = 2U,`

`kClockClkoutSelLpoClk = 3U,`

`kClockClkoutSelMcgIrClk = 4U,`

`kClockClkoutSelOsc0erClk = 6U }`

SIM CLKOUT_SEL clock source select.
- enum `sim_adc_pretrg_sel_kv10z7_t` {

`kSimAdcPretrgselA,`

`kSimAdcPretrgselB }`

SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_kv10z7_t` {

SIM HAL driver

- ```
kSimAdcTrgSelExt = 0U,
kSimAdcTrgSelHighSpeedComp0 = 1U,
kSimAdcTrgSelHighSpeedComp1 = 2U,
kSimAdcTrgSelDma0 = 4U,
kSimAdcTrgSelDma1 = 5U,
kSimAdcTrgSelDma2 = 6U,
kSimAdcTrgSelDma3 = 7U,
kSimAdcTrgSelFtm0 = 8U,
kSimAdcTrgSelFtm1 = 9U,
kSimAdcTrgSelFtm2 = 10U,
kSimAdcTrgSelLptimer = 14U }
```
- SIM ADCx trigger select.*
- enum `sim_uart_rxsr_kv10z7_t` {  
    kSimUartRxsrPin,  
    kSimUartRxsrCmp0,  
    kSimUartRxsrCmp1 }
- SIM receive data source select.*
- enum `sim_uart_txsr_kv10z7_t` {  
    kSimUartTxsrPin,  
    kSimUartTxsrFtm1,  
    kSimUartTxsrFtm2 }
- SIM transmit data source select.*
- enum `sim_ftm_trg_src_kv10z7_t` {  
    kSimFtmTrgSrc0,  
    kSimFtmTrgSrc1 }
- SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_kv10z7_t` {  
    kSimFtmClkSel0,  
    kSimFtmClkSel1,  
    kSimFtmClkSel2 }
- SIM FlexTimer external clock select.*
- enum `clock_ftm_fixedfreq_src_kv10z7_t` {  
    kClockFtmClkMcgFfClk = 0U,  
    kClockFtmClkMcgIrClk = 1U,  
    kClockFtmClkOsc0erClk = 2U }
- SIM FlexTimer Fixed Frequency clock source.*
- enum `clock_adc_alt_src_kv10z7_t` {  
    kClockAdcAltClkSrcOutdiv5 = 0U,  
    kClockAdcAltClkSrcMcgIrClk = 1U,  
    kClockAdcAltClkSrcOsc0erClk = 2U }
- SIM ADC alt clock source.*
- enum `sim_ftm_ch_src_kv10z7_t` {  
    kSimFtmChSrc0,  
    kSimFtmChSrc1,  
    kSimFtmChSrc2,  
    kSimFtmChSrc3 }

- *SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftm_ch_out_src_kv10z7_t` {
   
    `kSimFtmChOutSrc0,`
  
    `kSimFtmChOutSrc1 }`
  - *SIM FlexTimer x channel y output source select.*
  - enum `sim_ftm_ftl_sel_kv10z7_t` {
   
    `kSimFtmFtlSel0,`
  
    `kSimFtmFtlSel1 }`
    - *SIM FlexTimer x Fault y select.*
    - enum `sim_ftm_ftl_carrier_sel_kv10z7_t` {
   
    `kSimFtmCarrierSel0,`
  
    `kSimFtmCarrierSel1 }`
      - *SIM FlexTimer0/2 output channel Carrier frequency selection.*
      - enum `sim_clock_gate_name_kv10z7_t`
        - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

#### 40.2.15.2 Macro Definition Documentation

##### 40.2.15.2.1 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

#### 40.2.15.3 Enumeration Type Documentation

##### 40.2.15.3.1 enum clock\_wdog\_src\_kv10z7\_t

Enumerator

- kClockWdogSrcLpoClk* LPO.
- kClockWdogSrcAltClk* Alternative clock.

##### 40.2.15.3.2 enum clock\_lptmr\_src\_kv10z7\_t

Enumerator

- kClockLptmrSrcMcgIrClk* MCG IRC clock.
- kClockLptmrSrcLpoClk* LPO clock.
- kClockLptmrSrcEr32kClk* ERCLK32K clock.
- kClockLptmrSrcOsc0erClk* OSCERCLK clock.

##### 40.2.15.3.3 enum clock\_er32k\_src\_kv10z7\_t

Enumerator

- kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).
- kClockEr32kSrcLpo* LPO clock.

## SIM HAL driver

### 40.2.15.3.4 enum clock\_clkout\_src\_kv10z7\_t

Enumerator

- kClockClkoutSelBusClk* Bus clock.
- kClockClkoutSelLpoClk* LPO clock.
- kClockClkoutSelMcgIrClk* MCG IRC clock.
- kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 40.2.15.3.5 enum sim\_adc\_pretrg\_sel\_kv10z7\_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 40.2.15.3.6 enum sim\_adc\_trg\_sel\_kv10z7\_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelDma0* DMA channel 0.
- kSimAdcTrgSelDma1* DMA channel 1.
- kSimAdcTrgSelDma2* DMA channel 2.
- kSimAdcTrgSelDma3* DMA channel 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

### 40.2.15.3.7 enum sim\_uart\_rxsrc\_kv10z7\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

**40.2.15.3.8 enum sim\_uart\_txsrc\_kv10z7\_t**

Enumerator

*kSimUartTxsrcPin* UARTx\_TX Pin.*kSimUartTxsrcFtm1* UARTx\_TX pin modulated with FTM1 channel 0 output.*kSimUartTxsrcFtm2* UARTx\_TX pin modulated with FTM2 channel 0 output.**40.2.15.3.9 enum sim\_ftm\_trg\_src\_kv10z7\_t**

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.**40.2.15.3.10 enum sim\_ftm\_clk\_sel\_kv10z7\_t**

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.*kSimFtmClkSel1* FTM CLKIN1 pin.*kSimFtmClkSel2* FTM CLKIN2 pin.**40.2.15.3.11 enum clock\_ftm\_fixedfreq\_src\_kv10z7\_t**

Enumerator

*kClockFtmClkMcgFfClk* MCGFFCLK.*kClockFtmClkMcgIrClk* MCGIRCLK.*kClockFtmClkOsc0erClk* OSCERCLK.**40.2.15.3.12 enum clock\_adc\_alt\_src\_kv10z7\_t**

Enumerator

*kClockAdcAltClkSrcOutdiv5* OUTDIV5 output clock.*kClockAdcAltClkSrcMcgIrClk* MCGIRCLK.*kClockAdcAltClkSrcOsc0erClk* OSCERCLK.

## SIM HAL driver

### 40.2.15.3.13 enum sim\_ftm\_ch\_src\_kv10z7\_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y uses input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y uses input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y uses input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y uses input capture source 3.

### 40.2.15.3.14 enum sim\_ftm\_ch\_out\_src\_kv10z7\_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source 1.

### 40.2.15.3.15 enum sim\_ftm\_flt\_sel\_kv10z7\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 40.2.15.3.16 enum sim\_ftm\_flt\_carrier\_sel\_kv10z7\_t

Enumerator

- kSimFtmCarrierSel0* Carrier frequency selection 0.
- kSimFtmCarrierSel1* Carrier frequency selection 1.

### 40.2.15.3.17 enum sim\_clock\_gate\_name\_kv10z7\_t

## 40.2.16 K60D10 SIM HAL driver

### 40.2.16.1 Overview

The section describes the enumerations, macros and data structures for K60D10 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK60D10.h](#)

#### Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k60d10\\_t](#) {
   
    kClockWdogSrcLpoClk,
   
    kClockWdogSrcAltClk }
   
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k60d10\\_t](#) {
   
    kClockTraceSrcMcgoutClk,
   
    kClockTraceSrcCoreClk }
   
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k60d10\\_t](#) {
   
    kClockPortFilterSrcBusClk,
   
    kClockPortFilterSrcLpoClk }
   
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k60d10\\_t](#) {
   
    kClockLptmrSrcMcgIrClk,
   
    kClockLptmrSrcLpoClk,
   
    kClockLptmrSrcEr32kClk,
   
    kClockLptmrSrcOsc0erClk }
   
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k60d10\\_t](#) {
   
    kClockTimeSrcCoreSysClk,
   
    kClockTimeSrcPllFllSel,
   
    kClockTimeSrcOsc0erClk,
   
    kClockTimeSrcExt }
   
*SIM timestamp clock source.*
- enum [clock\\_rmii\\_src\\_k60d10\\_t](#) {
   
    kClockRmiiSrcExtalClk,
   
    kClockRmiiSrcExt }
   
*SIM RMII clock source.*

## SIM HAL driver

- enum `clock_usbfs_src_k60d10_t` {  
  `kClockUsbfsSrcExt`,  
  `kClockUsbfsSrcPllFllSel` }  
    *SIM USB FS clock source.*
- enum `clock_flexcan_src_k60d10_t` {  
  `kClockFlexcanSrcOsc0erClk`,  
  `kClockFlexcanSrcBusClk` }  
    *FLEXCAN clock source select.*
- enum `clock_sdhc_src_k60d10_t` {  
  `kClockSdhcSrcCoreSysClk`,  
  `kClockSdhcSrcPllFllSel`,  
  `kClockSdhcSrcOsc0erClk`,  
  `kClockSdhcSrcExt` }  
    *SDHC clock source.*
- enum `clock_sai_src_k60d10_t` {  
  `kClockSaiSrcSysClk` = 0U,  
  `kClockSaiSrcOsc0erClk` = 1U,  
  `kClockSaiSrcPllClk` = 3U }  
    *SAI clock source.*
- enum `clock_tsi_active_mode_src_k60d10_t` {  
  `kClockTsiActiveSrcBusClk`,  
  `kClockTsiActiveSrcMcgIrClk`,  
  `kClockTsiActiveSrcOsc0erClk` }  
    *TSI Active Mode clock source.*
- enum `clock_tsi_lp_mode_src_k60d10_t` {  
  `kClockTsiLpSrcLpoClk`,  
  `kClockTsiLpSrcEr32kClk` }  
    *TSI Low-power Mode clock source.*
- enum `clock_pllfl_sel_k60d10_t` {  
  `kClockPllFllSelFll` = 0U,  
  `kClockPllFllSelPll` = 1U }  
    *SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k60d10_t` {  
  `kClockEr32kSrcOsc0` = 0U,  
  `kClockEr32kSrcRtc` = 2U,  
  `kClockEr32kSrcLpo` = 3U }  
    *SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k60d10_t` {  
  `kClockClkoutSelFlexbusClk` = 0U,  
  `kClockClkoutSelFlashClk` = 2U,  
  `kClockClkoutSelLpoClk` = 3U,  
  `kClockClkoutSelMcgIrClk` = 4U,  
  `kClockClkoutSelRtc32kClk` = 5U,  
  `kClockClkoutSelOsc0erClk` = 6U }  
    *SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_k60d10_t` {  
  `kClockRtcoutSrc1Hz`,

- kClockRtcoutSrc32kHz }
 

*SIM RTCCLKOUTSEL clock source select.*
- enum sim\_usbsstby\_mode\_k60d10\_t {
 

kSimUsbstbyNoRegulator,  
kSimUsbstbyWithRegulator }

*SIM USB voltage regulator in standby mode setting during stop modes.*
- enum sim\_usbvstby\_mode\_k60d10\_t {
 

kSimUsvstbyNoRegulator,  
kSimUsvstbyWithRegulator }

*SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum sim\_adc\_pretrg\_sel\_k60d10\_t {
 

kSimAdcPretrgselA,  
kSimAdcPretrgselB }

*SIM ADCx pre-trigger select.*
- enum sim\_adc\_trg\_sel\_k60d10\_t {
 

kSimAdcTrgselExt = 0U,  
kSimAdcTrgSelHighSpeedComp0 = 1U,  
kSimAdcTrgSelHighSpeedComp1 = 2U,  
kSimAdcTrgSelHighSpeedComp2 = 3U,  
kSimAdcTrgSelPit0 = 4U,  
kSimAdcTrgSelPit1 = 5U,  
kSimAdcTrgSelPit2 = 6U,  
kSimAdcTrgSelPit3 = 7U,  
kSimAdcTrgSelFtm0 = 8U,  
kSimAdcTrgSelFtm1 = 9U,  
kSimAdcTrgSelFtm2 = 10U,  
kSimAdcTrgSelRtcAlarm = 12U,  
kSimAdcTrgSelRtcSec = 13U,  
kSimAdcTrgSelLptimer = 14U }

*SIM ADCx trigger select.*
- enum sim\_uart\_rxsra\_k60d10\_t {
 

kSimUartRxsraPin,  
kSimUartRxsraCmp0,  
kSimUartRxsraCmp1 }

*SIM receive data source select.*
- enum sim\_uart\_txsrc\_k60d10\_t {
 

kSimUartTxsrcPin,  
kSimUartTxsrcFtm1,  
kSimUartTxsrcFtm2 }

*SIM transmit data source select.*
- enum sim\_ftm\_trg\_src\_k60d10\_t {
 

kSimFtmTrgSrc0,  
kSimFtmTrgSrc1 }

*SIM FlexTimer x trigger y select.*
- enum sim\_ftm\_clk\_sel\_k60d10\_t {
 

kSimFtmClkSel0,  
kSimFtmClkSel1 }

## SIM HAL driver

- *SIM FlexTimer external clock select.*  
• enum `sim_ftm_ch_src_k60d10_t` {  
  kSimFtmChSrc0,  
  kSimFtmChSrc1,  
  kSimFtmChSrc2,  
  kSimFtmChSrc3 }
- *SIM FlexTimer x channel y input capture source select.*  
• enum `sim_ftm_ftl_sel_k60d10_t` {  
  kSimFtmFltSel0,  
  kSimFtmFltSel1 }
- *SIM FlexTimer x Fault y select.*  
• enum `sim_tpm_clk_sel_k60d10_t` {  
  kSimTpmClkSel0,  
  kSimTpmClkSel1 }
- *SIM Timer/PWM external clock select.*  
• enum `sim_tpm_ch_src_k60d10_t` {  
  kSimTpmChSrc0,  
  kSimTpmChSrc1 }
- *SIM Timer/PWM x channel y input capture source select.*  
• enum `sim_cmtuartpad_streng_k60d10_t` {  
  kSimCmtuartSinglePad,  
  kSimCmtuartDualPad }
- *SIM CMT/UART pad drive strength.*  
• enum `sim_ptd7pad_streng_k60d10_t` {  
  kSimPtd7padSinglePad,  
  kSimPtd7padDualPad }
- *SIM PTD7 pad drive strength.*  
• enum `sim_flexport_security_level_k60d10_t` {  
  kSimFbslLevel0,  
  kSimFbslLevel1,  
  kSimFbslLevel2,  
  kSimFbslLevel3 }
- *SIM FlexBus security level.*  
• enum `sim_clock_gate_name_k60d10_t`  
  *Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 40.2.16.2 Macro Definition Documentation

#### 40.2.16.2.1 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

### 40.2.16.3 Enumeration Type Documentation

#### 40.2.16.3.1 enum clock\_wdog\_src\_k60d10\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for K60D10 it is Bus clock.

#### 40.2.16.3.2 enum clock\_trace\_src\_k60d10\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 40.2.16.3.3 enum clock\_port\_filter\_src\_k60d10\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 40.2.16.3.4 enum clock\_lptmr\_src\_k60d10\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 40.2.16.3.5 enum clock\_time\_src\_k60d10\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.

*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].

*kClockTimeSrcOsc0erClk* OSCERCLK clock.

*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

#### 40.2.16.3.6 enum clock\_rmii\_src\_k60d10\_t

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.

*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

## SIM HAL driver

### 40.2.16.3.7 enum clock\_usbfs\_src\_k60d10\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

### 40.2.16.3.8 enum clock\_flexcan\_src\_k60d10\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.

*kClockFlexcanSrcBusClk* Bus clock.

### 40.2.16.3.9 enum clock\_sdhc\_src\_k60d10\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.

*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].

*kClockSdhcSrcOsc0erClk* OSCERCLK clock.

*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 40.2.16.3.10 enum clock\_sai\_src\_k60d10\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.

*kClockSaiSrcOsc0erClk* OSC0ERCLK.

*kClockSaiSrcPllClk* MCGPLLCLK.

### 40.2.16.3.11 enum clock\_tsi\_active\_mode\_src\_k60d10\_t

Enumerator

*kClockTsiActiveSrcBusClk* Bus clock.

*kClockTsiActiveSrcMcgIrcClk* MCG IRC clock.

*kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

### 40.2.16.3.12 enum clock\_tsi\_lp\_mode\_src\_k60d10\_t

Enumerator

*kClockTsiLpSrcLpoClk* LPO clock.

*kClockTsiLpSrcEr32kClk* ERCLK32K clock.

**40.2.16.3.13 enum clock\_pllfill\_sel\_k60d10\_t**

Enumerator

- kClockPllFillSelFll* Fll clock.
- kClockPllFillSelPll* Pll0 clock.

**40.2.16.3.14 enum clock\_er32k\_src\_k60d10\_t**

Enumerator

- kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).
- kClockEr32kSrcRtc* RTC 32k clock .
- kClockEr32kSrcLpo* LPO clock.

**40.2.16.3.15 enum clock\_clkout\_src\_k60d10\_t**

Enumerator

- kClockClkoutSelFlexbusClk* Flexbus clock.
- kClockClkoutSelFlashClk* Flash clock.
- kClockClkoutSelLpoClk* LPO clock.
- kClockClkoutSelMcgIrClk* MCG out clock.
- kClockClkoutSelRtc32kClk* RTC 32k clock.
- kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

**40.2.16.3.16 enum clock\_rtcout\_src\_k60d10\_t**

Enumerator

- kClockRtcoutSrc1Hz* 1Hz clock
- kClockRtcoutSrc32kHz* 32kHz clock

**40.2.16.3.17 enum sim\_usbsstby\_mode\_k60d10\_t**

Enumerator

- kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes
- kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

## SIM HAL driver

### 40.2.16.3.18 enum sim\_usbvstby\_mode\_k60d10\_t

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes

*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 40.2.16.3.19 enum sim\_adc\_pretrg\_sel\_k60d10\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.

*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 40.2.16.3.20 enum sim\_adc\_trg\_sel\_k60d10\_t

Enumerator

*kSimAdcTrgselExt* External trigger.

*kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.

*kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.

*kSimAdcTrgSelHighSpeedComp2* High speed comparator 2 output.

*kSimAdcTrgSelPit0* PIT trigger 0.

*kSimAdcTrgSelPit1* PIT trigger 1.

*kSimAdcTrgSelPit2* PIT trigger 2.

*kSimAdcTrgSelPit3* PIT trigger 3.

*kSimAdcTrgSelFtm0* FTM0 trigger.

*kSimAdcTrgSelFtm1* FTM1 trigger.

*kSimAdcTrgSelFtm2* FTM2 trigger.

*kSimAdcTrgSelRtcAlarm* RTC alarm.

*kSimAdcTrgSelRtcSec* RTC seconds.

*kSimAdcTrgSelLptimer* Low-power timer trigger.

### 40.2.16.3.21 enum sim\_uart\_rxsrv\_k60d10\_t

Enumerator

*kSimUartRxsrvPin* UARTx\_RX Pin.

*kSimUartRxsrvCmp0* CMP0.

*kSimUartRxsrvCmp1* CMP1.

**40.2.16.3.22 enum sim\_uart\_txsrc\_k60d10\_t**

Enumerator

*kSimUartTxsrcPin* UARTx\_TX Pin.*kSimUartTxsrcFtm1* UARTx\_TX pin modulated with FTM1 channel 0 output.*kSimUartTxsrcFtm2* UARTx\_TX pin modulated with FTM2 channel 0 output.**40.2.16.3.23 enum sim\_ftm\_trg\_src\_k60d10\_t**

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.**40.2.16.3.24 enum sim\_ftm\_clk\_sel\_k60d10\_t**

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.*kSimFtmClkSel1* FTM CLKIN1 pin.**40.2.16.3.25 enum sim\_ftm\_ch\_src\_k60d10\_t**

Enumerator

*kSimFtmChSrc0* FlexTimer x channel y input capture source 0.*kSimFtmChSrc1* FlexTimer x channel y input capture source 1.*kSimFtmChSrc2* FlexTimer x channel y input capture source 2.*kSimFtmChSrc3* FlexTimer x channel y input capture source 3.**40.2.16.3.26 enum sim\_ftm\_flt\_sel\_k60d10\_t**

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.*kSimFtmFltSel1* FlexTimer x fault y select 1.**40.2.16.3.27 enum sim\_tpm\_clk\_sel\_k60d10\_t**

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

## SIM HAL driver

### 40.2.16.3.28 enum sim\_tpm\_ch\_src\_k60d10\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.

*kSimTpmChSrc1* CMP0 output.

### 40.2.16.3.29 enum sim\_cmtuartpad\_strenght\_k60d10\_t

Enumerator

*kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.

*kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

### 40.2.16.3.30 enum sim\_ptd7pad\_strenght\_k60d10\_t

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.

*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 40.2.16.3.31 enum sim\_flexport\_security\_level\_k60d10\_t

Enumerator

*kSimFlexLevel0* FlexBus security level 0.

*kSimFlexLevel1* FlexBus security level 1.

*kSimFlexLevel2* FlexBus security level 2.

*kSimFlexLevel3* FlexBus security level 3.

### 40.2.16.3.32 enum sim\_clock\_gate\_name\_k60d10\_t

## 40.2.17 KV30F12810 SIM HAL driver

### 40.2.17.1 Overview

The section describes the enumerations, macros and data structures for KV30F12810 SIM HAL driver.

#### Files

- file `fsl_sim_hal_MKV30F12810.h`

#### Macros

- #define `FSL_SIM_SCGC_BIT(SCGCx, n)` (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum `clock_wdog_src_kv30f12810_t` {
   
`kClockWdogSrcLpoClk,`
  
`kClockWdogSrcAltClk }`
  
*WDOG clock source select.*
- enum `clock_trace_src_kv30f12810_t` {
   
`kClockTraceSrcMcgoutClk,`
  
`kClockTraceSrcCoreClk }`
  
*Debug trace clock source select.*
- enum `clock_port_filter_src_kv30f12810_t` {
   
`kClockPortFilterSrcBusClk,`
  
`kClockPortFilterSrcLpoClk }`
  
*PORTx digital input filter clock source select.*
- enum `clock_lptmr_src_kv30f12810_t` {
   
`kClockLptmrSrcMcgIrClk,`
  
`kClockLptmrSrcLpoClk,`
  
`kClockLptmrSrcEr32kClk,`
  
`kClockLptmrSrcOsc0erClkUndiv }`
  
*LPTMR clock source select.*
- enum `clock_pllfl_sel_kv30f12810_t` {
   
`kClockPliflSelFll = 0U,`
  
`kClockPliflSelIrc48M = 3U }`
  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kv30f12810_t` {
   
`kClockEr32kSrcOsc0 = 0U,`
  
`kClockEr32kSrcLpo = 3U }`
  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_kv30f12810_t` {

## SIM HAL driver

- ```
kClockClkoutSelFlashClk = 2U,  
kClockClkoutSelLpoClk = 3U,  
kClockClkoutSelMcgIrClk = 4U,  
kClockClkoutSelOsc0erClk = 6U,  
kClockClkoutSelIrc48M = 7U }  
    SIM CLKOUT_SEL clock source select.  
• enum clock\_osc32kout\_sel\_kv30f12810\_t {  
    kClockOsc32koutNone = 0U,  
    kClockOsc32koutPte0 = 1U,  
    kClockOsc32koutPte26 = 2U }  
    SIM OSC32KOUT selection.  
• enum sim\_adc\_pretrg\_sel\_kv30f12810\_t {  
    kSimAdcPretrgselA,  
    kSimAdcPretrgselB }  
    SIM ADCx pre-trigger select.  
• enum sim\_adc\_trg\_sel\_kv30f12810\_t {  
    kSimAdcTrgselExt = 0U,  
    kSimAdcTrgSelHighSpeedComp0 = 1U,  
    kSimAdcTrgSelHighSpeedComp1 = 2U,  
    kSimAdcTrgSelPit0 = 4U,  
    kSimAdcTrgSelPit1 = 5U,  
    kSimAdcTrgSelPit2 = 6U,  
    kSimAdcTrgSelPit3 = 7U,  
    kSimAdcTrgSelFtm0 = 8U,  
    kSimAdcTrgSelFtm1 = 9U,  
    kSimAdcTrgSelFtm2 = 10U,  
    kSimAdcTrgSelLptimer = 14U }  
    SIM ADCx trigger select.  
• enum sim\_uart\_rxsrc\_kv30f12810\_t {  
    kSimUartRxsrcPin,  
    kSimUartRxsrcCmp0,  
    kSimUartRxsrcCmp1 }  
    SIM receive data source select.  
• enum sim\_uart\_txsrc\_kv30f12810\_t {  
    kSimUartTxsrcPin,  
    kSimUartTxsrcFtm1,  
    kSimUartTxsrcFtm2 }  
    SIM transmit data source select.  
• enum sim\_ftm\_trg\_src\_kv30f12810\_t {  
    kSimFtmTrgSrc0,  
    kSimFtmTrgSrc1 }  
    SIM FlexTimer x trigger y select.  
• enum sim\_ftm\_clk\_sel\_kv30f12810\_t {  
    kSimFtmClkSel0,  
    kSimFtmClkSel1 }  
    SIM FlexTimer external clock select.  
• enum sim\_ftm\_ch\_src\_kv30f12810\_t {
```

```
kSimFtmChSrc0,
kSimFtmChSrc1,
kSimFtmChSrc2,
kSimFtmChSrc3 }
```

SIM FlexTimer x channel y input capture source select.

- enum `sim_ftm_ch_out_src_kv30f12810_t` {
 kSimFtmChOutSrc0,
 kSimFtmChOutSrc1 }

SIM FlexTimer x channel y output source select.

- enum `sim_ftm_flt_sel_kv30f12810_t` {
 kSimFtmFltSel0,
 kSimFtmFltSel1 }

SIM FlexTimer x Fault y select.

- enum `sim_clock_gate_name_kv30f12810_t`

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

40.2.17.2 Macro Definition Documentation

40.2.17.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.17.3 Enumeration Type Documentation

40.2.17.3.1 enum clock_wdog_src_kv30f12810_t

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

40.2.17.3.2 enum clock_trace_src_kv30f12810_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.17.3.3 enum clock_port_filter_src_kv30f12810_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

SIM HAL driver

40.2.17.3.4 enum clock_lptmr_src_kv30f12810_t

Enumerator

- kClockLptmrSrcMcgIrClk* MCGIRCLK.
- kClockLptmrSrcLpoClk* LPO clock.
- kClockLptmrSrcEr32kClk* ERCLK32K clock.
- kClockLptmrSrcOsc0erClkUndiv* OSCERCLK_UNDIV clock.

40.2.17.3.5 enum clock_pllfll_sel_kv30f12810_t

Enumerator

- kClockPllFllSelFll* Fll clock.
- kClockPllFllSelIrc48M* IRC48MCLK.

40.2.17.3.6 enum clock_er32k_src_kv30f12810_t

Enumerator

- kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).
- kClockEr32kSrcLpo* LPO clock.

40.2.17.3.7 enum clock_clkout_src_kv30f12810_t

Enumerator

- kClockClkoutSelFlashClk* Flash clock.
- kClockClkoutSelLpoClk* LPO clock.
- kClockClkoutSelMcgIrClk* MCGIRCLK.
- kClockClkoutSelOsc0erClk* OSC0ERCLK.
- kClockClkoutSelIrc48M* IRC48MCLK.

40.2.17.3.8 enum clock_osc32kout_sel_kv30f12810_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

40.2.17.3.9 enum sim_adc_pretrg_sel_kv30f12810_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.**40.2.17.3.10 enum sim_adc_trg_sel_kv30f12810_t**

Enumerator

kSimAdcTrgselExt External trigger.*kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.*kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.*kSimAdcTrgSelPit0* PIT trigger 0.*kSimAdcTrgSelPit1* PIT trigger 1.*kSimAdcTrgSelPit2* PIT trigger 2.*kSimAdcTrgSelPit3* PIT trigger 3.*kSimAdcTrgSelFtm0* FTM0 trigger.*kSimAdcTrgSelFtm1* FTM1 trigger.*kSimAdcTrgSelFtm2* FTM2 trigger.*kSimAdcTrgSelLptimer* Low-power timer trigger.**40.2.17.3.11 enum sim_uart_rxsrv_kv30f12810_t**

Enumerator

kSimUartRxsrvPin UARTx_RX Pin.*kSimUartRxsrvCmp0* CMP0.*kSimUartRxsrvCmp1* CMP1.**40.2.17.3.12 enum sim_uart_txsrc_kv30f12810_t**

Enumerator

kSimUartTxsrcPin UARTx_TX Pin.*kSimUartTxsrcFtm1* UARTx_TX pin modulated with FTM1 channel 0 output.*kSimUartTxsrcFtm2* UARTx_TX pin modulated with FTM2 channel 0 output.**40.2.17.3.13 enum sim_ftm_trg_src_kv30f12810_t**

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

SIM HAL driver

40.2.17.3.14 enum sim_ftm_clk_sel_kv30f12810_t

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

40.2.17.3.15 enum sim_ftm_ch_src_kv30f12810_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

40.2.17.3.16 enum sim_ftm_ch_out_src_kv30f12810_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source 1.

40.2.17.3.17 enum sim_ftm_flt_sel_kv30f12810_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

40.2.17.3.18 enum sim_clock_gate_name_kv30f12810_t

40.2.18 KV31F12810 SIM HAL driver

40.2.18.1 Overview

The section describes the enumerations, macros and data structures for KV31F12810 SIM HAL driver.

Files

- file `fsl_sim_hal_MKV31F12810.h`

Macros

- #define `FSL_SIM_SCGC_BIT(SCGCx, n)` (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum `clock_wdog_src_kv31f12810_t` {

`kClockWdogSrcLpoClk,`

`kClockWdogSrcAltClk }`

WDOG clock source select.
- enum `clock_trace_src_kv31f12810_t` {

`kClockTraceSrcMcgoutClk,`

`kClockTraceSrcCoreClk }`

Debug trace clock source select.
- enum `clock_port_filter_src_kv31f12810_t` {

`kClockPortFilterSrcBusClk,`

`kClockPortFilterSrcLpoClk }`

PORTx digital input filter clock source select.
- enum `clock_lptmr_src_kv31f12810_t` {

`kClockLptmrSrcMcgIrClk,`

`kClockLptmrSrcLpoClk,`

`kClockLptmrSrcEr32kClk,`

`kClockLptmrSrcOsc0erClkUndiv }`

LPTMR clock source select.
- enum `clock_lpuart_src_kv31f12810_t`

SIM LPUART clock source.
- enum `clock_pllfl_sel_kv31f12810_t` {

`kClockPliflSelFl1 = 0U,`

`kClockPliflSelIrc48M = 3U }`

SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_kv31f12810_t` {

`kClockEr32kSrcOsc0 = 0U,`

`kClockEr32kSrcLpo = 3U }`

SIM external reference clock source select (OSC32KSEL).

SIM HAL driver

- enum `clock_clkout_src_kv31f12810_t` {
 `kClockClkoutSelFlashClk` = 2U,
 `kClockClkoutSelLpoClk` = 3U,
 `kClockClkoutSelMcgIrClk` = 4U,
 `kClockClkoutSelOsc0erClk` = 6U,
 `kClockClkoutSelIrc48M` = 7U }
- SIM CLKOUT_SEL clock source select.
- enum `clock_osc32kout_sel_kv31f12810_t` {
 `kClockOsc32koutNone` = 0U,
 `kClockOsc32koutPte0` = 1U,
 `kClockOsc32koutPte26` = 2U }
- SIM OSC32KOUT selection.
- enum `sim_adc_pretrg_sel_kv31f12810_t` {
 `kSimAdcPretrgselA`,
 `kSimAdcPretrgselB` }
- SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_kv31f12810_t` {
 `kSimAdcTrgselExt` = 0U,
 `kSimAdcTrgSelHighSpeedComp0` = 1U,
 `kSimAdcTrgSelHighSpeedComp1` = 2U,
 `kSimAdcTrgSelPit0` = 4U,
 `kSimAdcTrgSelPit1` = 5U,
 `kSimAdcTrgSelPit2` = 6U,
 `kSimAdcTrgSelPit3` = 7U,
 `kSimAdcTrgSelFtm0` = 8U,
 `kSimAdcTrgSelFtm1` = 9U,
 `kSimAdcTrgSelFtm2` = 10U,
 `kSimAdcTrgSelLptimer` = 14U }
- SIM ADCx trigger select.
- enum `sim_lpuart_rxsrv_kv31f12810_t` {
 `kSimLpuartRxsrvPin`,
 `kSimLpuartRxsrvCmp0`,
 `kSimLpuartRxsrvCmp1` }
- SIM LPUART RX source.
- enum `sim_uart_rxsrv_kv31f12810_t` {
 `kSimUartRxsrvPin`,
 `kSimUartRxsrvCmp0`,
 `kSimUartRxsrvCmp1` }
- SIM receive data source select.
- enum `sim_uart_txsrc_kv31f12810_t` {
 `kSimUartTxsrcPin`,
 `kSimUartTxsrcFtm1`,
 `kSimUartTxsrcFtm2` }
- SIM transmit data source select.
- enum `sim_ftm_trg_src_kv31f12810_t` {
 `kSimFtmTrgSrc0`,

- `kSimFtmTrgSrc1 }`
`SIM FlexTimer x trigger y select.`
- enum `sim_ftm_clk_sel_kv31f12810_t` {
 `kSimFtmClkSel0,`
`kSimFtmClkSel1 }`
`SIM FlexTimer external clock select.`
- enum `sim_ftm_ch_src_kv31f12810_t` {
 `kSimFtmChSrc0,`
`kSimFtmChSrc1,`
`kSimFtmChSrc2,`
`kSimFtmChSrc3 }`
`SIM FlexTimer x channel y input capture source select.`
- enum `sim_ftm_ch_out_src_kv31f12810_t` {
 `kSimFtmChOutSrc0,`
`kSimFtmChOutSrc1 }`
`SIM FlexTimer x channel y output source select.`
- enum `sim_ftm_flt_sel_kv31f12810_t` {
 `kSimFtmFltSel0,`
`kSimFtmFltSel1 }`
`SIM FlexTimer x Fault y select.`
- enum `sim_clock_gate_name_kv31f12810_t`
`Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.`

40.2.18.2 Macro Definition Documentation

40.2.18.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.18.3 Enumeration Type Documentation

40.2.18.3.1 enum clock_wdog_src_kv31f12810_t

Enumerator

`kClockWdogSrcLpoClk` LPO.

`kClockWdogSrcAltClk` Alternative clock, for this SOC it is Bus clock.

40.2.18.3.2 enum clock_trace_src_kv31f12810_t

Enumerator

`kClockTraceSrcMcgoutClk` MCG out clock.

`kClockTraceSrcCoreClk` core clock

SIM HAL driver

40.2.18.3.3 enum clock_port_filter_src_kv31f12810_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.18.3.4 enum clock_lptmr_src_kv31f12810_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

kClockLptmrSrcLpoClk LPO clock.

kClockLptmrSrcEr32kClk ERCLK32K clock.

kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

40.2.18.3.5 enum clock_pllfl_sel_kv31f12810_t

Enumerator

kClockPllFllSelFll Fll clock.

kClockPllFllSelIrc48M IRC48MCLK.

40.2.18.3.6 enum clock_er32k_src_kv31f12810_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).

kClockEr32kSrcLpo LPO clock.

40.2.18.3.7 enum clock_clkout_src_kv31f12810_t

Enumerator

kClockClkoutSelFlashClk Flash clock.

kClockClkoutSelLpoClk LPO clock.

kClockClkoutSelMcgIrClk MCGIRCLK.

kClockClkoutSelOsc0erClk OSC0ERCLK.

kClockClkoutSelIrc48M IRC48MCLK.

40.2.18.3.8 enum clock_osc32kout_sel_kv31f12810_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

40.2.18.3.9 enum sim_adc_pretrg_sel_kv31f12810_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

40.2.18.3.10 enum sim_adc_trg_sel_kv31f12810_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

40.2.18.3.11 enum sim_lpuart_rxsrv_kv31f12810_t

Enumerator

- kSimLpuartRxsrvPin* LPUARTx_RX Pin.
- kSimLpuartRxsrvCmp0* CMP0.
- kSimLpuartRxsrvCmp1* CMP1.

40.2.18.3.12 enum sim_uart_rxsrv_kv31f12810_t

Enumerator

- kSimUartRxsrvPin* UARTx_RX Pin.

SIM HAL driver

kSimUartRxsrCmp0 CMP0.
kSimUartRxsrCmp1 CMP1.

40.2.18.3.13 enum sim_uart_txsrc_kv31f12810_t

Enumerator

kSimUartTxsrcPin UARTx_TX Pin.
kSimUartTxsrcFtm1 UARTx_TX pin modulated with FTM1 channel 0 output.
kSimUartTxsrcFtm2 UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.18.3.14 enum sim_ftm_trg_src_kv31f12810_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

40.2.18.3.15 enum sim_ftm_clk_sel_kv31f12810_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

40.2.18.3.16 enum sim_ftm_ch_src_kv31f12810_t

Enumerator

kSimFtmChSrc0 FlexTimer x channel y input capture source 0.
kSimFtmChSrc1 FlexTimer x channel y input capture source 1.
kSimFtmChSrc2 FlexTimer x channel y input capture source 2.
kSimFtmChSrc3 FlexTimer x channel y input capture source 3.

40.2.18.3.17 enum sim_ftm_ch_out_src_kv31f12810_t

Enumerator

kSimFtmChOutSrc0 FlexTimer x channel y output source 0.
kSimFtmChOutSrc1 FlexTimer x channel y output source 1.

40.2.18.3.18 enum sim_ftm_ftl_sel_kv31f12810_t

Enumerator

kSimFtmFltSel0 FlexTimer x fault y select 0.

kSimFtmFltSel1 FlexTimer x fault y select 1.

40.2.18.3.19 enum sim_clock_gate_name_kv31f12810_t

SIM HAL driver

40.2.19 KV31F25612 SIM HAL driver

40.2.19.1 Overview

The section describes the enumerations, macros and data structures for KV31F25612 SIM HAL driver.

Files

- file [fsl_sim_hal_MKV31F25612.h](#)

Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_kv31f25612_t](#) {
 kClockWdogSrcLpoClk,
 kClockWdogSrcAltClk }
WDOG clock source select.
- enum [clock_trace_src_kv31f25612_t](#) {
 kClockTraceSrcMcgoutClk,
 kClockTraceSrcCoreClk }
Debug trace clock source select.
- enum [clock_port_filter_src_kv31f25612_t](#) {
 kClockPortFilterSrcBusClk,
 kClockPortFilterSrcLpoClk }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_kv31f25612_t](#) {
 kClockLptmrSrcMcgIrClk,
 kClockLptmrSrcLpoClk,
 kClockLptmrSrcEr32kClk,
 kClockLptmrSrcOsc0erClkUndiv }
LPTMR clock source select.
- enum [clock_lpuart_src_kv31f25612_t](#) {
 kClockLpuartSrcNone,
 kClockLpuartSrcPllFllSel,
 kClockLpuartSrcOsc0erClk,
 kClockLpuartSrcMcgIrClk }
SIM LPUART clock source.
- enum [clock_pllfl_sel_kv31f25612_t](#) {
 kClockPllFllSelFll = 0U,
 kClockPllFllSelPll = 1U,
 kClockPllFllSelIrc48M = 3U }

- *SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kv31f25612_t` {

 `kClockEr32kSrcOsc0` = 0U,

 `kClockEr32kSrcLpo` = 3U }
- *SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_kv31f25612_t` {

 `kClockClkoutSelFlashClk` = 2U,

 `kClockClkoutSelLpoClk` = 3U,

 `kClockClkoutSelMcgIrClk` = 4U,

 `kClockClkoutSelOsc0erClk` = 6U,

 `kClockClkoutSelIrc48M` = 7U }
- *SIM CLKOUT_SEL clock source select.*
- enum `clock_osc32kout_sel_kv31f25612_t` {

 `kClockOsc32koutNone` = 0U,

 `kClockOsc32koutPte0` = 1U,

 `kClockOsc32koutPte26` = 2U }
- *SIM OSC32KOUT selection.*
- enum `sim_adc_pretrg_sel_kv31f25612_t` {

 `kSimAdcPretrgselA`,

 `kSimAdcPretrgselB` }
- *SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kv31f25612_t` {

 `kSimAdcTrgselExt` = 0U,

 `kSimAdcTrgSelHighSpeedComp0` = 1U,

 `kSimAdcTrgSelHighSpeedComp1` = 2U,

 `kSimAdcTrgSelPit0` = 4U,

 `kSimAdcTrgSelPit1` = 5U,

 `kSimAdcTrgSelPit2` = 6U,

 `kSimAdcTrgSelPit3` = 7U,

 `kSimAdcTrgSelFtm0` = 8U,

 `kSimAdcTrgSelFtm1` = 9U,

 `kSimAdcTrgSelFtm2` = 10U,

 `kSimAdcTrgSelLptimer` = 14U }
- *SIM ADCx trigger select.*
- enum `sim_lpuart_rxsrv_kv31f25612_t` {

 `kSimLpuartRxsrvPin`,

 `kSimLpuartRxsrvCmp0`,

 `kSimLpuartRxsrvCmp1` }
- *SIM LPUART RX source.*
- enum `sim_uart_rxsrv_kv31f25612_t` {

 `kSimUartRxsrvPin`,

 `kSimUartRxsrvCmp0`,

 `kSimUartRxsrvCmp1` }
- *SIM receive data source select.*
- enum `sim_uart_txsrc_kv31f25612_t` {

 `kSimUartTxsrcPin`,

 `kSimUartTxsrcFtm1`,

SIM HAL driver

```
kSimUartTxsrcFtm2 }  
    SIM transmit data source select.  
• enum sim_ftm_trg_src_kv31f25612_t {  
    kSimFtmTrgSrc0,  
    kSimFtmTrgSrc1 }  
        SIM FlexTimer x trigger y select.  
• enum sim_ftm_clk_sel_kv31f25612_t {  
    kSimFtmClkSel0,  
    kSimFtmClkSel1 }  
        SIM FlexTimer external clock select.  
• enum sim_ftm_ch_src_kv31f25612_t {  
    kSimFtmChSrc0,  
    kSimFtmChSrc1,  
    kSimFtmChSrc2,  
    kSimFtmChSrc3 }  
        SIM FlexTimer x channel y input capture source select.  
• enum sim_ftm_ch_out_src_kv31f25612_t {  
    kSimFtmChOutSrc0,  
    kSimFtmChOutSrc1 }  
        SIM FlexTimer x channel y output source select.  
• enum sim_ftm_flt_sel_kv31f25612_t {  
    kSimFtmFltSel0,  
    kSimFtmFltSel1 }  
        SIM FlexTimer x Fault y select.  
• enum sim_clock_gate_name_kv31f25612_t  
    Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.
```

40.2.19.2 Macro Definition Documentation

40.2.19.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.19.3 Enumeration Type Documentation

40.2.19.3.1 enum clock_wdog_src_kv31f25612_t

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

40.2.19.3.2 enum clock_trace_src_kv31f25612_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.19.3.3 enum clock_port_filter_src_kv31f25612_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.*kClockPortFilterSrcLpoClk* LPO.**40.2.19.3.4 enum clock_lptmr_src_kv31f25612_t**

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.*kClockLptmrSrcLpoClk* LPO clock.*kClockLptmrSrcEr32kClk* ERCLK32K clock.*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK_UNDIV clock.**40.2.19.3.5 enum clock_lpuart_src_kv31f25612_t**

Enumerator

kClockLpuartSrcNone Clock disabled.*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLSEL].*kClockLpuartSrcOsc0erClk* OSCERCLK.*kClockLpuartSrcMcgIrClk* MCGIRCLK.**40.2.19.3.6 enum clock_pllfll_sel_kv31f25612_t**

Enumerator

kClockPllFllSelFll Fll clock.*kClockPllFllSelPll* Pll0 clock.*kClockPllFllSelIrc48M* IRC48MCLK.**40.2.19.3.7 enum clock_er32k_src_kv31f25612_t**

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).*kClockEr32kSrcLpo* LPO clock.

SIM HAL driver

40.2.19.3.8 enum clock_clkout_src_kv31f25612_t

Enumerator

- kClockClkoutSelFlashClk* Flash clock.
- kClockClkoutSelLpoClk* LPO clock.
- kClockClkoutSelMcgIrClk* MCGIRCLK.
- kClockClkoutSelOsc0erClk* OSC0ERCLK.
- kClockClkoutSelIrc48M* IRC48MCLK.

40.2.19.3.9 enum clock_osc32kout_sel_kv31f25612_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

40.2.19.3.10 enum sim_adc_pretrg_sel_kv31f25612_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

40.2.19.3.11 enum sim_adc_trg_sel_kv31f25612_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

40.2.19.3.12 enum sim_lpuart_rxsrv_kv31f25612_t

Enumerator

kSimLpuartRxsrvPin LPUARTx_RX Pin.

kSimLpuartRxsrvCmp0 CMP0.

kSimLpuartRxsrvCmp1 CMP1.

40.2.19.3.13 enum sim_uart_rxsrv_kv31f25612_t

Enumerator

kSimUartRxsrvPin UARTx_RX Pin.

kSimUartRxsrvCmp0 CMP0.

kSimUartRxsrvCmp1 CMP1.

40.2.19.3.14 enum sim_uart_txsrc_kv31f25612_t

Enumerator

kSimUartTxsrcPin UARTx_TX Pin.

kSimUartTxsrcFtm1 UARTx_TX pin modulated with FTM1 channel 0 output.

kSimUartTxsrcFtm2 UARTx_TX pin modulated with FTM2 channel 0 output.

40.2.19.3.15 enum sim_ftm_trg_src_kv31f25612_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.

kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

40.2.19.3.16 enum sim_ftm_clk_sel_kv31f25612_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.

kSimFtmClkSel1 FTM CLKIN1 pin.

SIM HAL driver

40.2.19.3.17 enum sim_ftm_ch_src_kv31f25612_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

40.2.19.3.18 enum sim_ftm_ch_out_src_kv31f25612_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source 1.

40.2.19.3.19 enum sim_ftm_flt_sel_kv31f25612_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

40.2.19.3.20 enum sim_clock_gate_name_kv31f25612_t

40.2.20 KV31F51212 SIM HAL driver

40.2.20.1 Overview

The section describes the enumerations, macros and data structures for KV31F51212 SIM HAL driver.

Files

- file `fsl_sim_hal_MKV31F51212.h`

Macros

- #define `FSL_SIM_SCGC_BIT(SCGCx, n)` (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum `clock_wdog_src_kv31f51212_t` {

`kClockWdogSrcLpoClk,`

`kClockWdogSrcAltClk }`

WDOG clock source select.
- enum `clock_trace_src_kv31f51212_t` {

`kClockTraceSrcMcgoutClk,`

`kClockTraceSrcCoreClk }`

Debug trace clock source select.
- enum `clock_port_filter_src_kv31f51212_t` {

`kClockPortFilterSrcBusClk,`

`kClockPortFilterSrcLpoClk }`

PORTx digital input filter clock source select.
- enum `clock_lptmr_src_kv31f51212_t` {

`kClockLptmrSrcMcgIrClk,`

`kClockLptmrSrcLpoClk,`

`kClockLptmrSrcEr32kClk,`

`kClockLptmrSrcOsc0erClkUndiv }`

LPTMR clock source select.
- enum `clock_lpuart_src_kv31f51212_t` {

`kClockLpuartSrcNone,`

`kClockLpuartSrcPllFllSel,`

`kClockLpuartSrcOsc0erClk,`

`kClockLpuartSrcMcgIrClk }`

SIM LPUART clock source.
- enum `clock_pllfl_sel_kv31f51212_t` {

`kClockPllFllSelFll = 0U,`

`kClockPllFllSelPll = 1U,`

`kClockPllFllSelIrc48M = 3U }`

SIM HAL driver

- *SIM PLLFLLSEL clock source select.*
 - enum `clock_er32k_src_kv31f51212_t` {
 `kClockEr32kSrcOsc0` = 0U,
 `kClockEr32kSrcLpo` = 3U }
- *SIM external reference clock source select (OSC32KSEL).*
 - enum `clock_clkout_src_kv31f51212_t` {
 `kClockClkoutSelFlexbusClk` = 0U,
 `kClockClkoutSelFlashClk` = 2U,
 `kClockClkoutSelLpoClk` = 3U,
 `kClockClkoutSelMcgIrClk` = 4U,
 `kClockClkoutSelOsc0erClk` = 6U,
 `kClockClkoutSelIrc48M` = 7U }
- *SIM CLKOUT_SEL clock source select.*
 - enum `clock_osc32kout_sel_kv31f51212_t` {
 SIM OSC32KOUT selection.
 - enum `sim_adc_pretrg_sel_kv31f51212_t` {
 `kSimAdcPretrgselA`,
 `kSimAdcPretrgselB` }
- *SIM ADCx pre-trigger select.*
 - enum `sim_adc_trg_sel_kv31f51212_t` {
 `kSimAdcTrgselExt` = 0U,
 `kSimAdcTrgSelHighSpeedComp0` = 1U,
 `kSimAdcTrgSelHighSpeedComp1` = 2U,
 `kSimAdcTrgSelPit0` = 4U,
 `kSimAdcTrgSelPit1` = 5U,
 `kSimAdcTrgSelPit2` = 6U,
 `kSimAdcTrgSelPit3` = 7U,
 `kSimAdcTrgSelFtm0` = 8U,
 `kSimAdcTrgSelFtm1` = 9U,
 `kSimAdcTrgSelFtm2` = 10U,
 `kSimAdcTrgSelFtm3` = 11U,
 `kSimAdcTrgSelLptimer` = 14U }
- *SIM ADCx trigger select.*
 - enum `sim_lpuart_rxsrv_kv31f51212_t` {
 `kSimLpuartRxsrvPin`,
 `kSimLpuartRxsrvCmp0`,
 `kSimLpuartRxsrvCmp1` }
- *SIM LPUART RX source.*
 - enum `sim_uart_rxsrv_kv31f51212_t` {
 `kSimUartRxsrvPin`,
 `kSimUartRxsrvCmp0`,
 `kSimUartRxsrvCmp1` }
- *SIM receive data source select.*
 - enum `sim_uart_txsrc_kv31f51212_t` {
 `kSimUartTxsrcPin`,
 `kSimUartTxsrcFtm1`,
 `kSimUartTxsrcFtm2` }

- *SIM transmit data source select.*
• enum `sim_ftm_trg_src_kv31f51212_t` {
 kSimFtmTrgSrc0,
 kSimFtmTrgSrc1 }
- *SIM FlexTimer x trigger y select.*
• enum `sim_ftm_clk_sel_kv31f51212_t` {
 kSimFtmClkSel0,
 kSimFtmClkSel1 }
- *SIM FlexTimer external clock select.*
• enum `sim_ftm_ch_src_kv31f51212_t` {
 kSimFtmChSrc0,
 kSimFtmChSrc1,
 kSimFtmChSrc2,
 kSimFtmChSrc3 }
- *SIM FlexTimer x channel y input capture source select.*
• enum `sim_ftm_ch_out_src_kv31f51212_t` {
 kSimFtmChOutSrc0,
 kSimFtmChOutSrc1 }
- *SIM FlexTimer x channel y output source select.*
• enum `sim_ftm_flt_sel_kv31f51212_t` {
 kSimFtmFltSel0,
 kSimFtmFltSel1 }
- *SIM FlexTimer x Fault y select.*
• enum `sim_flexport_security_level_kv31f51212_t` {
 kSimFbslLevel0,
 kSimFbslLevel1,
 kSimFbslLevel2,
 kSimFbslLevel3 }
- *SIM FlexBus security level.*
• enum `sim_clock_gate_name_kv31f51212_t`
 Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

IP related clock feature APIs

- static void `SIM_HAL_EnableClock` (uint32_t baseAddr, sim_clock_gate_name_t name)
 Enable the clock for specific module.
- static void `SIM_HAL_DisableClock` (uint32_t baseAddr, sim_clock_gate_name_t name)
 Disable the clock for specific module.
- static bool `SIM_HAL_GetGateCmd` (uint32_t baseAddr, sim_clock_gate_name_t name)
 Get the the clock gate state for specific module.
- static void `CLOCK_HAL_SetLpuartSrc` (uint32_t baseAddr, uint32_t instance, clock_lpuart_src_t setting)
 Set LPUART clock source.
- static clock_lpuart_src_t `CLOCK_HAL_GetLpuartSrc` (uint32_t baseAddr, uint32_t instance)
 Get LPUART clock source.
- static void `CLOCK_HAL_SetTraceClkSrc` (uint32_t baseAddr, clock_trace_src_t setting)
 Set debug trace clock selection.
- static clock_trace_src_t `CLOCK_HAL_GetTraceClkSrc` (uint32_t baseAddr)

SIM HAL driver

- static void **CLOCK_HAL_SetExternalRefClock32kSrc** (uint32_t baseAddr, clock_er32k_src_t setting)
 - Set the clock selection of ERCLK32K.
- static clock_er32k_src_t **CLOCK_HAL_GetExternalRefClock32kSrc** (uint32_t baseAddr)
 - Get the clock selection of ERCLK32K.
- static void **CLOCK_HAL_SetPllf1Sel** (uint32_t baseAddr, clock_pllfl_sel_t setting)
 - Set PLL/FLL clock selection.
- static clock_pllfl_sel_t **CLOCK_HAL_GetPllf1Sel** (uint32_t baseAddr)
 - Get PLL/FLL clock selection.
- static void **CLOCK_HAL_SetClkOutSel** (uint32_t baseAddr, clock_clkout_src_t setting)
 - Set CLKOUTSEL selection.
- static clock_clkout_src_t **CLOCK_HAL_GetClkOutSel** (uint32_t baseAddr)
 - Get CLKOUTSEL selection.
- static void **CLOCK_HAL_SetOsc32kOutSel** (uint32_t baseAddr, clock_osc32kout_sel_t setting)
 - Set OSC32KOUT selection.
- static clock_osc32kout_sel_t **CLOCK_HAL_GetOsc32kOutSel** (uint32_t baseAddr)
 - Get OSC32KOUT selection.
- static void **CLOCK_HAL_SetOutDiv1** (uint32_t baseAddr, uint8_t setting)
 - Set OUTDIV1.
- static uint8_t **CLOCK_HAL_GetOutDiv1** (uint32_t baseAddr)
 - Get OUTDIV1.
- static void **CLOCK_HAL_SetOutDiv2** (uint32_t baseAddr, uint8_t setting)
 - Set OUTDIV2.
- static uint8_t **CLOCK_HAL_GetOutDiv2** (uint32_t baseAddr)
 - Get OUTDIV2.
- static void **CLOCK_HAL_SetOutDiv3** (uint32_t baseAddr, uint8_t setting)
 - Set OUTDIV3.
- static uint8_t **CLOCK_HAL_GetOutDiv3** (uint32_t baseAddr)
 - Get OUTDIV3.
- static void **CLOCK_HAL_SetOutDiv4** (uint32_t baseAddr, uint8_t setting)
 - Set OUTDIV4.
- static uint8_t **CLOCK_HAL_GetOutDiv4** (uint32_t baseAddr)
 - Get OUTDIV4.
- void **CLOCK_HAL_SetOutDiv** (uint32_t baseAddr, uint8_t outdiv1, uint8_t outdiv2, uint8_t outdiv3, uint8_t outdiv4)
 - Sets the clock out dividers setting.
- void **CLOCK_HAL_GetOutDiv** (uint32_t baseAddr, uint8_t *outdiv1, uint8_t *outdiv2, uint8_t *outdiv3, uint8_t *outdiv4)
 - Gets the clock out dividers setting.
- static uint32_t **SIM_HAL_GetRamSize** (uint32_t baseAddr)
 - Gets RAM size.
- static void **SIM_HAL_SetFlexbusSecurityLevelMode** (uint32_t baseAddr, sim_flexbus_security_level_t setting)
 - Sets the FlexBus security level setting.
- static sim_flexbus_security_level_t **SIM_HAL_GetFlexbusSecurityLevelMode** (uint32_t baseAddr)
 - Gets the FlexBus security level setting.
- void **SIM_HAL_SetAdcAlternativeTriggerCmd** (uint32_t baseAddr, uint32_t instance, bool enable)
 - Sets the ADCx alternate trigger enable setting.
- bool **SIM_HAL_GetAdcAlternativeTriggerCmd** (uint32_t baseAddr, uint32_t instance)
 - Gets the ADCx alternate trigger enable setting.

- void **SIM_HAL_SetAdcPreTriggerMode** (uint32_t baseAddr, uint32_t instance, sim_adc_pretrg_sel_t select)

Sets the ADCx pre-trigger select setting.
- sim_adc_pretrg_sel_t **SIM_HAL_GetAdcPreTriggerMode** (uint32_t baseAddr, uint32_t instance)

Gets the ADCx pre-trigger select setting.
- void **SIM_HAL_SetAdcTriggerMode** (uint32_t baseAddr, uint32_t instance, sim_adc_trg_sel_t select)

Sets the ADCx trigger select setting.
- sim_adc_trg_sel_t **SIM_HAL_GetAdcTriggerMode** (uint32_t baseAddr, uint32_t instance)

Gets the ADCx trigger select setting.
- void **SIM_HAL_SetAdcTriggerModeOneStep** (uint32_t baseAddr, uint32_t instance, bool altTrigEn, sim_adc_pretrg_sel_t preTrigSel, sim_adc_trg_sel_t trigSel)

Sets the ADCx trigger select setting in one function.
- void **SIM_HAL_SetUartRxSrcMode** (uint32_t baseAddr, uint32_t instance, sim_uart_rxsrv_t select)

Sets the UARTx receive data source select setting.
- sim_uart_rxsrv_t **SIM_HAL_GetUartRxSrcMode** (uint32_t baseAddr, uint32_t instance)

Gets the UARTx receive data source select setting.
- void **SIM_HAL_SetUartTxSrcMode** (uint32_t baseAddr, uint32_t instance, sim_uart_txsrc_t select)

Sets the UARTx transmit data source select setting.
- sim_uart_txsrc_t **SIM_HAL_GetUartTxSrcMode** (uint32_t baseAddr, uint32_t instance)

Gets the UARTx transmit data source select setting.
- static void **SIM_HAL_SetLpuartRxSrcMode** (uint32_t baseAddr, uint32_t instance, sim_lpuart_rxsrv_t select)

Sets the LPUARTx receive data source select setting.
- static sim_lpuart_rxsrv_t **SIM_HAL_GetLpuartRxSrcMode** (uint32_t baseAddr, uint32_t instance)

Gets the LPUARTx receive data source select setting.
- void **SIM_HAL_SetFtmTriggerSrcMode** (uint32_t baseAddr, uint32_t instance, uint8_t trigger, sim_ftm_trg_src_t select)

Sets the FlexTimer x hardware trigger y source select setting.
- sim_ftm_trg_src_t **SIM_HAL_GetFtmTriggerSrcMode** (uint32_t baseAddr, uint32_t instance, uint8_t trigger)

Gets the FlexTimer x hardware trigger y source select setting.
- void **SIM_HAL_SetFtmExternalClkPinMode** (uint32_t baseAddr, uint32_t instance, sim_ftm_clk_sel_t select)

Sets the FlexTimer x external clock pin select setting.
- sim_ftm_clk_sel_t **SIM_HAL_GetFtmExternalClkPinMode** (uint32_t baseAddr, uint32_t instance)

Gets the FlexTimer x external clock pin select setting.
- void **SIM_HAL_SetFtmChSrcMode** (uint32_t baseAddr, uint32_t instance, uint8_t channel, sim_ftm_ch_src_t select)

Sets the FlexTimer x channel y input capture source select setting.
- sim_ftm_ch_src_t **SIM_HAL_GetFtmChSrcMode** (uint32_t baseAddr, uint32_t instance, uint8_t channel)

Gets the FlexTimer x channel y input capture source select setting.
- void **SIM_HAL_SetFtmChOutSrcMode** (uint32_t baseAddr, uint32_t instance, uint8_t channel, sim_ftm_ch_out_src_t select)

Sets the FlexTimer x channel y output source select setting.
- sim_ftm_ch_out_src_t **SIM_HAL_GetFtmChOutSrcMode** (uint32_t baseAddr, uint32_t instance, uint8_t channel)

Gets the FlexTimer x channel y output source select setting.

SIM HAL driver

- Gets the FlexTimer x channel y output source select setting.
• void **SIM_HAL_SetFtmSyncCmd** (uint32_t baseAddr, uint32_t instance, bool sync)
 Set FlexTimer x hardware trigger 0 software synchronization.
- static bool **SIM_HAL_GetFtmSyncCmd** (uint32_t baseAddr, uint32_t instance)
 Get FlexTimer x hardware trigger 0 software synchronization setting.
- void **SIM_HAL_SetFtmFaultSelMode** (uint32_t baseAddr, uint32_t instance, uint8_t fault, **sim_ftm_ftl_sel_t** select)
 Sets the FlexTimer x fault y select setting.
- **sim_ftm_ftl_sel_t SIM_HAL_GetFtmFaultSelMode** (uint32_t baseAddr, uint32_t instance, uint8_t fault)
 Gets the FlexTimer x fault y select setting.
- static uint32_t **SIM_HAL_GetFamilyId** (uint32_t baseAddr)
 Gets the Kinetis Family ID in the System Device ID register (SIM_SDID).
- static uint32_t **SIM_HAL_GetSubFamilyId** (uint32_t baseAddr)
 Gets the Kinetis Sub-Family ID in the System Device ID register (SIM_SDID).
- static uint32_t **SIM_HAL_GetSeriesId** (uint32_t baseAddr)
 Gets the Kinetis SeriesID in the System Device ID register (SIM_SDID).
- static uint32_t **SIM_HAL_GetPinCntId** (uint32_t baseAddr)
 Gets the Kinetis Pincount ID in System Device ID register (SIM_SDID).
- static uint32_t **SIM_HAL_GetRevId** (uint32_t baseAddr)
 Gets the Kinetis Revision ID in the System Device ID register (SIM_SDID).
- static uint32_t **SIM_HAL_GetDieId** (uint32_t baseAddr)
 Gets the Kinetis Die ID in the System Device ID register (SIM_SDID).
- static uint32_t **SIM_HAL_GetProgramFlashSize** (uint32_t baseAddr)
 Gets the program flash size in the Flash Configuration Register 1 (SIM_FCFG).
- static void **SIM_HAL_SetFlashDoze** (uint32_t baseAddr, uint32_t setting)
 Sets the Flash Doze in the Flash Configuration Register 1 (SIM_FCFG).
- static uint32_t **SIM_HAL_GetFlashDoze** (uint32_t baseAddr)
 Gets the Flash Doze in the Flash Configuration Register 1 (SIM_FCFG).
- static void **SIM_HAL_SetFlashDisableCmd** (uint32_t baseAddr, bool disable)
 Sets the Flash disable setting.
- static bool **SIM_HAL_GetFlashDisableCmd** (uint32_t baseAddr)
 Gets the Flash disable setting.
- static uint32_t **SIM_HAL_GetFlashMaxAddrBlock0** (uint32_t baseAddr)
 Gets the Flash maximum address block 0 in the Flash Configuration Register 1 (SIM_FCFG).
- static uint32_t **SIM_HAL_GetFlashMaxAddrBlock1** (uint32_t baseAddr)
 Gets the Flash maximum address block 1 in Flash Configuration Register 2.

40.2.20.2 Macro Definition Documentation

40.2.20.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

40.2.20.3 Enumeration Type Documentation

40.2.20.3.1 enum clock_wdog_src_kv31f51212_t

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

40.2.20.3.2 enum clock_trace_src_kv31f51212_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

40.2.20.3.3 enum clock_port_filter_src_kv31f51212_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

40.2.20.3.4 enum clock_lptmr_src_kv31f51212_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.

kClockLptmrSrcLpoClk LPO clock.

kClockLptmrSrcEr32kClk ERCLK32K clock.

kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

40.2.20.3.5 enum clock_lpuart_src_kv31f51212_t

Enumerator

kClockLpuartSrcNone Clock disabled.

kClockLpuartSrcPllFllSel Clock as selected by SOPT2[PLLFLSEL].

kClockLpuartSrcOsc0erClk OSCERCLK.

kClockLpuartSrcMcgIrClk MCGIRCLK.

40.2.20.3.6 enum clock_pllfll_sel_kv31f51212_t

Enumerator

kClockPllFllSelFll Fll clock.

kClockPllFllSelPll Pll0 clock.

kClockPllFllSelIrc48M IRC48MCLK.

SIM HAL driver

40.2.20.3.7 enum clock_er32k_src_kv31f51212_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).

kClockEr32kSrcLpo LPO clock.

40.2.20.3.8 enum clock_clkout_src_kv31f51212_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.

kClockClkoutSelFlashClk Flash clock.

kClockClkoutSelLpoClk LPO clock.

kClockClkoutSelMcgIrClk MCGIRCLK.

kClockClkoutSelOsc0erClk OSC0ERCLK.

kClockClkoutSelIrc48M IRC48MCLK.

40.2.20.3.9 enum sim_adc_pretrg_sel_kv31f51212_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.

kSimAdcPretrgselB Pre-trigger B selected for ADCx.

40.2.20.3.10 enum sim_adc_trg_sel_kv31f51212_t

Enumerator

kSimAdcTrgselExt External trigger.

kSimAdcTrgSelHighSpeedComp0 High speed comparator 0 output.

kSimAdcTrgSelHighSpeedComp1 High speed comparator 1 output.

kSimAdcTrgSelPit0 PIT trigger 0.

kSimAdcTrgSelPit1 PIT trigger 1.

kSimAdcTrgSelPit2 PIT trigger 2.

kSimAdcTrgSelPit3 PIT trigger 3.

kSimAdcTrgSelFtm0 FTM0 trigger.

kSimAdcTrgSelFtm1 FTM1 trigger.

kSimAdcTrgSelFtm2 FTM2 trigger.

kSimAdcTrgSelFtm3 FTM3 trigger.

kSimAdcTrgSelLptimer Low-power timer trigger.

40.2.20.3.11 enum sim_lpuart_rxsrv_kv31f51212_t

Enumerator

kSimLpuartRxsrvPin LPUARTx_RX Pin.*kSimLpuartRxsrvCmp0* CMP0.*kSimLpuartRxsrvCmp1* CMP1.**40.2.20.3.12 enum sim_uart_rxsrv_kv31f51212_t**

Enumerator

kSimUartRxsrvPin UARTx_RX Pin.*kSimUartRxsrvCmp0* CMP0.*kSimUartRxsrvCmp1* CMP1.**40.2.20.3.13 enum sim_uart_txsrc_kv31f51212_t**

Enumerator

kSimUartTxsrcPin UARTx_TX Pin.*kSimUartTxsrcFtm1* UARTx_TX pin modulated with FTM1 channel 0 output.*kSimUartTxsrcFtm2* UARTx_TX pin modulated with FTM2 channel 0 output.**40.2.20.3.14 enum sim_ftm_trg_src_kv31f51212_t**

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.**40.2.20.3.15 enum sim_ftm_clk_sel_kv31f51212_t**

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.*kSimFtmClkSel1* FTM CLKIN1 pin.

SIM HAL driver

40.2.20.3.16 enum sim_ftm_ch_src_kv31f51212_t

Enumerator

- kSimFtmChSrc0* FlexTimer x channel y input capture source 0.
- kSimFtmChSrc1* FlexTimer x channel y input capture source 1.
- kSimFtmChSrc2* FlexTimer x channel y input capture source 2.
- kSimFtmChSrc3* FlexTimer x channel y input capture source 3.

40.2.20.3.17 enum sim_ftm_ch_out_src_kv31f51212_t

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source 1.

40.2.20.3.18 enum sim_ftm_flt_sel_kv31f51212_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

40.2.20.3.19 enum sim_flexbus_security_level_kv31f51212_t

Enumerator

- kSimFbslLevel0* FlexBus security level 0.
- kSimFbslLevel1* FlexBus security level 1.
- kSimFbslLevel2* FlexBus security level 2.
- kSimFbslLevel3* FlexBus security level 3.

40.2.20.3.20 enum sim_clock_gate_name_kv31f51212_t

40.2.20.4 Function Documentation

40.2.20.4.1 static void SIM_HAL_EnableClock(uint32_t baseAddr, sim_clock_gate_name_t name) [inline], [static]

This function enables the clock for specific module.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>name</i>	Name of the module to enable.

40.2.20.4.2 static void SIM_HAL_DisableClock (*uint32_t baseAddr*, *sim_clock_gate_name_t name*) [inline], [static]

This function disables the clock for specific module.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>name</i>	Name of the module to disable.

40.2.20.4.3 static bool SIM_HAL_GetGateCmd (*uint32_t baseAddr*, *sim_clock_gate_name_t name*) [inline], [static]

This function will get the clock gate state for specific module.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>name</i>	Name of the module to get.

Returns

state true - ungated(Enabled), false - gated (Disabled)

40.2.20.4.4 static void CLOCK_HAL_SetLpuartSrc (*uint32_t baseAddr*, *uint32_t instance*, *clock_lpuart_src_t setting*) [inline], [static]

This function sets lpuart clock source selection.

Parameters

SIM HAL driver

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	LPUART instance.
<i>setting</i>	The value to set.

40.2.20.4.5 static clock_lpuart_src_t CLOCK_HAL_GetLpuartSrc (uint32_t *baseAddr*, uint32_t *instance*) [inline], [static]

This function gets lpuart clock source selection.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	LPUART instance.

Returns

Current selection.

40.2.20.4.6 static void CLOCK_HAL_SetTraceClkSrc (uint32_t *baseAddr*, clock_trace_src_t *setting*) [inline], [static]

This function sets debug trace clock selection.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.7 static clock_trace_src_t CLOCK_HAL_GetTraceClkSrc (uint32_t *baseAddr*) [inline], [static]

This function gets debug trace clock selection.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current selection.

**40.2.20.4.8 static void CLOCK_HAL_SetExternalRefClock32kSrc (uint32_t *baseAddr*,
clock_er32k_src_t *setting*) [inline], [static]**

This function sets the clock selection of ERCLK32K.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.9 static clock_er32k_src_t CLOCK_HAL_GetExternalRefClock32kSrc (uint32_t baseAddr) [inline], [static]

This function gets the clock selection of ERCLK32K.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current selection.

40.2.20.4.10 static void CLOCK_HAL_SetPllfllSel (uint32_t baseAddr, clock_pllfill_sel_t setting) [inline], [static]

This function sets the selection of the high frequency clock for various peripheral clocking options

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.11 static clock_pllfill_sel_t CLOCK_HAL_GetPllfllSel (uint32_t baseAddr) [inline], [static]

This function gets the selection of the high frequency clock for various peripheral clocking options

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current selection.

40.2.20.4.12 static void CLOCK_HAL_SetClkOutSel (uint32_t *baseAddr*, clock_clkout_src_t *setting*) [inline], [static]

This function sets the selection of the clock to output on the CLKOUT pin.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.13 static clock_clkout_src_t CLOCK_HAL_GetClkOutSel (uint32_t *baseAddr*) [inline], [static]

This function gets the selection of the clock to output on the CLKOUT pin.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current selection.

40.2.20.4.14 static void CLOCK_HAL_SetOsc32kOutSel (uint32_t *baseAddr*, clock_osc32kout_sel_t *setting*) [inline], [static]

This function sets ERCLK32K output pin.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.15 static clock_osc32kout_sel_t CLOCK_HAL_GetOsc32kOutSel (uint32_t *baseAddr*) [inline], [static]

This function gets ERCLK32K output pin setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current selection.

**40.2.20.4.16 static void CLOCK_HAL_SetOutDiv1 (uint32_t *baseAddr*, uint8_t *setting*)
[inline], [static]**

This function sets divide value OUTDIV1.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.17 static uint8_t CLOCK_HAL_GetOutDiv1 (uint32_t *baseAddr*) [inline], [static]

This function gets divide value OUTDIV1.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current divide value.

40.2.20.4.18 static void CLOCK_HAL_SetOutDiv2 (uint32_t *baseAddr*, uint8_t *setting*) [inline], [static]

This function sets divide value OUTDIV2.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.19 static uint8_t CLOCK_HAL_GetOutDiv2 (uint32_t *baseAddr*) [inline], [static]

This function gets divide value OUTDIV2.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current divide value.

**40.2.20.4.20 static void CLOCK_HAL_SetOutDiv3 (uint32_t *baseAddr*, uint8_t *setting*)
[inline], [static]**

This function sets divide value OUTDIV3.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.21 static uint8_t CLOCK_HAL_GetOutDiv3 (uint32_t *baseAddr*) [inline], [static]

This function gets divide value OUTDIV3.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current divide value.

40.2.20.4.22 static void CLOCK_HAL_SetOutDiv4 (uint32_t *baseAddr*, uint8_t *setting*) [inline], [static]

This function sets divide value OUTDIV4.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

40.2.20.4.23 static uint8_t CLOCK_HAL_GetOutDiv4 (uint32_t *baseAddr*) [inline], [static]

This function gets divide value OUTDIV4.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

Current divide value.

**40.2.20.4.24 void CLOCK_HAL_SetOutDiv (uint32_t *baseAddr*, uint8_t *outdiv1*, uint8_t *outdiv2*,
uint8_t *outdiv3*, uint8_t *outdiv4*)**

This function sets the setting for all clock out dividers at the same time.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>outdiv1</i>	Outdivider1 setting
<i>outdiv2</i>	Outdivider2 setting
<i>outdiv3</i>	Outdivider3 setting
<i>outdiv4</i>	Outdivider4 setting

40.2.20.4.25 void CLOCK_HAL_GetOutDiv (uint32_t *baseAddr*, uint8_t * *outdiv1*, uint8_t * *outdiv2*, uint8_t * *outdiv3*, uint8_t * *outdiv4*)

This function gets the setting for all clock out dividers at the same time.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>outdiv1</i>	Outdivider1 setting
<i>outdiv2</i>	Outdivider2 setting
<i>outdiv3</i>	Outdivider3 setting
<i>outdiv4</i>	Outdivider4 setting

40.2.20.4.26 static uint32_t SIM_HAL_GetRamSize (uint32_t *baseAddr*) [inline], [static]

This function gets the RAM size. The field specifies the amount of system RAM available on the device.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

size RAM size on the device

40.2.20.4.27 static void SIM_HAL_SetFlexbusSecurityLevelMode (uint32_t *baseAddr*, sim_flexport_security_level_t *setting*) [inline], [static]

This function sets the FlexBus security level setting. If the security is enabled, this field affects which CPU operations can access the off-chip via the FlexBus and DDR controller interfaces. This field has no effect if the security is not enabled.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	FlexBus security level setting <ul style="list-style-type: none"> • 00: All off-chip accesses (op code and data) via the FlexBus and DDR controller are disallowed. • 10: Off-chip op code accesses are disallowed. Data accesses are allowed. • 11: Off-chip op code accesses and data accesses are allowed.

40.2.20.4.28 **static sim_flexport_security_level_t SIM_HAL_GetFlexbusSecurityLevelMode (uint32_t baseAddr) [inline], [static]**

This function gets the FlexBus security level setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

setting FlexBus security level setting

40.2.20.4.29 **void SIM_HAL_SetAdcAlternativeTriggerCmd (uint32_t baseAddr, uint32_t instance, bool enable) [inline]**

This function enables/disables the alternative conversion triggers for ADCx.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>enable</i>	Enable alternative conversion triggers for ADCx <ul style="list-style-type: none"> • true: Select alternative conversion trigger. • false: Select PDB trigger.

40.2.20.4.30 **bool SIM_HAL_GetAdcAlternativeTriggerCmd (uint32_t baseAddr, uint32_t instance) [inline]**

This function gets the ADCx alternate trigger enable setting.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

enabled True if ADCx alternate trigger is enabled

40.2.20.4.31 void SIM_HAL_SetAdcPreTriggerMode (uint32_t *baseAddr*, uint32_t *instance*, sim_adc_pretrg_sel_t *select*) [inline]

This function selects the ADCx pre-trigger source when the alternative triggers are enabled through ADCxALTTRGEN.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	pre-trigger select setting for ADCx

40.2.20.4.32 sim_adc_pretrg_sel_t SIM_HAL_GetAdcPreTriggerMode (uint32_t *baseAddr*, uint32_t *instance*) [inline]

This function gets the ADCx pre-trigger select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select ADCx pre-trigger select setting

40.2.20.4.33 void SIM_HAL_SetAdcTriggerMode (uint32_t *baseAddr*, uint32_t *instance*, sim_adc_trg_sel_t *select*) [inline]

This function selects the ADCx trigger source when alternative triggers are enabled through ADCxALTTRGEN.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	trigger select setting for ADCx

40.2.20.4.34 sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode (uint32_t *baseAddr*, uint32_t *instance*) [inline]

This function gets the ADCx trigger select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

ADCx trigger select setting

40.2.20.4.35 void SIM_HAL_SetAdcTriggerModeOneStep (uint32_t *baseAddr*, uint32_t *instance*, bool *altTrigEn*, sim_adc_pretrg_sel_t *preTrigSel*, sim_adc_trg_sel_t *trigSel*)

This function sets ADC alternate trigger, pre-trigger mode and trigger mode.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>altTrigEn</i>	Alternative trigger enable or not.
<i>preTrigSel</i>	Pre-trigger mode.
<i>trigSel</i>	Trigger mode.

40.2.20.4.36 void SIM_HAL_SetUartRxSrcMode (uint32_t *baseAddr*, uint32_t *instance*, sim_uart_rxsrc_t *select*)

This function selects the source for the UARTx receive data.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	the source for the UARTx receive data

40.2.20.4.37 sim_uart_rxsrc_t SIM_HAL_GetUartRxSrcMode (uint32_t *baseAddr*, uint32_t *instance*)

This function gets the UARTx receive data source select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select UARTx receive data source select setting

40.2.20.4.38 void SIM_HAL_SetUartTxSrcMode (uint32_t *baseAddr*, uint32_t *instance*, sim_uart_txsrc_t *select*)

This function selects the source for the UARTx transmit data.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	the source for the UARTx transmit data

40.2.20.4.39 sim_uart_txsrc_t SIM_HAL_GetUartTxSrcMode (uint32_t *baseAddr*, uint32_t *instance*)

This function gets the UARTx transmit data source select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select UARTx transmit data source select setting

40.2.20.4.40 static void SIM_HAL_SetLpuartRxSrcMode (uint32_t *baseAddr*, uint32_t *instance*, sim_lpuart_rxsrc_t *select*) [inline], [static]

This function selects the source for the LPUARTx receive data.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	the source for the LPUARTx receive data

40.2.20.4.41 static sim_lpuart_rxsrc_t SIM_HAL_GetLpuartRxSrcMode (uint32_t *baseAddr*, uint32_t *instance*) [inline], [static]

This function gets the LPUARTx receive data source select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select LPUARTx receive data source select setting

40.2.20.4.42 void SIM_HAL_SetFtmTriggerSrcMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *trigger*, sim_ftm_trg_src_t *select*)

This function selects the source of FTmx hardware trigger y.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>trigger</i>	hardware trigger y
<i>select</i>	FlexTimer x hardware trigger y <ul style="list-style-type: none">• 0: Pre-trigger A selected for ADCx.• 1: Pre-trigger B selected for ADCx.

40.2.20.4.43 sim_ftm_trg_src_t SIM_HAL_GetFtmTriggerSrcMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *trigger*)

This function gets the FlexTimer x hardware trigger y source select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>trigger</i>	hardware trigger y

Returns

select FlexTimer x hardware trigger y source select setting

40.2.20.4.44 void SIM_HAL_SetFtmExternalClkPinMode (uint32_t *baseAddr*, uint32_t *instance*, sim_ftm_clk_sel_t *select*)

This function selects the source of FTMx external clock pin select.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	FTMx external clock pin select <ul style="list-style-type: none">• 0: FTMx external clock driven by FTM CLKIN0 pin.• 1: FTMx external clock driven by FTM CLKIN1 pin.

**40.2.20.4.45 sim_ftm_clk_sel_t SIM_HAL_GetFtmExternalClkPinMode (uint32_t baseAddr,
 uint32_t instance)**

This function gets the FlexTimer x external clock pin select setting.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select FlexTimer x external clock pin select setting

40.2.20.4.46 void SIM_HAL_SetFtmChSrcMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *channel*, sim_ftm_ch_src_t *select*)

This function selects the FlexTimer x channel y input capture source.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>channel</i>	FlexTimer channel y
<i>select</i>	FlexTimer x channel y input capture source

40.2.20.4.47 sim_ftm_ch_src_t SIM_HAL_GetFtmChSrcMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *channel*)

This function gets the FlexTimer x channel y input capture source select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>channel</i>	FlexTimer channel y

Returns

select FlexTimer x channel y input capture source select setting

40.2.20.4.48 void SIM_HAL_SetFtmChOutSrcMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *channel*, sim_ftm_ch_out_src_t *select*)

This function selects the FlexTimer x channel y output source.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>channel</i>	FlexTimer channel y
<i>select</i>	FlexTimer x channel y output source

40.2.20.4.49 **sim_ftm_ch_out_src_t SIM_HAL_GetFtmChOutSrcMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *channel*)**

This function gets the FlexTimer x channel y output source select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>channel</i>	FlexTimer channel y

Returns

select FlexTimer x channel y output source select setting

40.2.20.4.50 **void SIM_HAL_SetFtmSyncCmd (uint32_t *baseAddr*, uint32_t *instance*, bool *sync*)**

This function sets FlexTimer x hardware trigger 0 software synchronization. FTMXSYNCBIT.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>sync</i>	Synchronize or not.

40.2.20.4.51 **static bool SIM_HAL_GetFtmSyncCmd (uint32_t *baseAddr*, uint32_t *instance*) [inline], [static]**

This function gets FlexTimer x hardware trigger 0 software synchronization. FTMXSYNCBIT.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

40.2.20.4.52 void SIM_HAL_SetFtmFaultSelMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *fault*, sim_ftm_ftl_sel_t *select*)

This function sets the FlexTimer x fault y select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>fault</i>	fault y
<i>select</i>	FlexTimer x fault y select setting <ul style="list-style-type: none">• 0: FlexTimer x fault y select 0.• 1: FlexTimer x fault y select 1.

40.2.20.4.53 sim_ftm_ftl_sel_t SIM_HAL_GetFtmFaultSelMode (uint32_t *baseAddr*, uint32_t *instance*, uint8_t *fault*)

This function gets the FlexTimer x fault y select setting.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>fault</i>	fault y

Returns

select FlexTimer x fault y select setting

40.2.20.4.54 static uint32_t SIM_HAL_GetFamilyId (uint32_t *baseAddr*) [inline], [static]

This function gets the Kinetis Family ID in the System Device ID register.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

id Kinetis Family ID

40.2.20.4.55 static uint32_t SIM_HAL_GetSubFamilyId (uint32_t *baseAddr*) [inline], [static]

This function gets the Kinetis Sub-Family ID in System Device ID register.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

id Kinetis Sub-Family ID

40.2.20.4.56 static uint32_t SIM_HAL_GetSeriesId (uint32_t *baseAddr*) [inline], [static]

This function gets the Kinetis Series ID in System Device ID register.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

id Kinetis Series ID

40.2.20.4.57 static uint32_t SIM_HAL_GetPinCntId (uint32_t *baseAddr*) [inline], [static]

This function gets the Kinetis Pincount ID in System Device ID register.

SIM HAL driver

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

id Kinetis Pincount ID

40.2.20.4.58 static uint32_t SIM_HAL_GetRevId(uint32_t *baseAddr*) [inline], [static]

This function gets the Kinetis Revision ID in System Device ID register.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

id Kinetis Revision ID

40.2.20.4.59 static uint32_t SIM_HAL_GetDieId(uint32_t *baseAddr*) [inline], [static]

This function gets the Kinetis Die ID in System Device ID register.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

id Kinetis Die ID

40.2.20.4.60 static uint32_t SIM_HAL_GetProgramFlashSize(uint32_t *baseAddr*) [inline], [static]

This function gets the program flash size in the Flash Configuration Register 1.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

size Program flash Size

40.2.20.4.61 static void SIM_HAL_SetFlashDoze (uint32_t *baseAddr*, uint32_t *setting*) [inline], [static]

This function sets the Flash Doze in the Flash Configuration Register 1.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
<i>setting</i>	Flash Doze setting

40.2.20.4.62 static uint32_t SIM_HAL_GetFlashDoze (uint32_t *baseAddr*) [inline], [static]

This function gets the Flash Doze in the Flash Configuration Register 1.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

setting Flash Doze setting

40.2.20.4.63 static void SIM_HAL_SetFlashDisableCmd (uint32_t *baseAddr*, bool *disable*) [inline], [static]

This function sets the Flash disable setting in the Flash Configuration Register 1.

Parameters

SIM HAL driver

<i>baseAddr</i>	Base address for current SIM instance.
<i>disable</i>	Flash disable setting

40.2.20.4.64 static bool SIM_HAL_GetFlashDisableCmd (uint32_t *baseAddr*) [inline], [static]

This function gets the Flash disable setting in the Flash Configuration Register 1.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

setting Flash disable setting

40.2.20.4.65 static uint32_t SIM_HAL_GetFlashMaxAddrBlock0 (uint32_t *baseAddr*) [inline], [static]

This function gets the Flash maximum block 0 in Flash Configuration Register 2.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

address Flash maximum block 0 address

40.2.20.4.66 static uint32_t SIM_HAL_GetFlashMaxAddrBlock1 (uint32_t *baseAddr*) [inline], [static]

This function gets the Flash maximum block 1 in Flash Configuration Register 1.

Parameters

<i>baseAddr</i>	Base address for current SIM instance.
-----------------	--

Returns

address Flash maximum block 0 address

40.2.21 K64F12 SIM HAL driver

40.2.21.1 Overview

The section describes the enumerations, macros and data structures for K64F12 SIM HAL driver.

Files

- file [fsl_sim_hal_MK64F12.h](#)

Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k64f12_t](#) {

 kClockWdogSrcLpoClk,

 kClockWdogSrcAltClk }

WDOG clock source select.
- enum [clock_trace_src_k64f12_t](#) {

 kClockTraceSrcMcgoutClk,

 kClockTraceSrcCoreClk }

Debug trace clock source select.
- enum [clock_port_filter_src_k64f12_t](#) {

 kClockPortFilterSrcBusClk,

 kClockPortFilterSrcLpoClk }

PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k64f12_t](#) {

 kClockLptmrSrcMcgIrClk,

 kClockLptmrSrcLpoClk,

 kClockLptmrSrcEr32kClk,

 kClockLptmrSrcOsc0erClk }

LPTMR clock source select.
- enum [clock_time_src_k64f12_t](#) {

 kClockTimeSrcCoreSysClk,

 kClockTimeSrcPllFllSel,

 kClockTimeSrcOsc0erClk,

 kClockTimeSrcExt }

SIM timestamp clock source.
- enum [clock_rmii_src_k64f12_t](#) {

 kClockRmiiSrcExtalClk,

 kClockRmiiSrcExt }

SIM RMII clock source.

SIM HAL driver

- enum `clock_usbfs_src_k64f12_t`{
 `kClockUsbfsSrcExt,`
 `kClockUsbfsSrcPllFllSel }`
 SIM USB FS clock source.
- enum `clock_flexcan_src_k64f12_t`{
 `kClockFlexcanSrcOsc0erClk,`
 `kClockFlexcanSrcBusClk }`
 FLEXCAN clock source select.
- enum `clock_sdhc_src_k64f12_t`{
 `kClockSdhcSrcCoreSysClk,`
 `kClockSdhcSrcPllFllSel,`
 `kClockSdhcSrcOsc0erClk,`
 `kClockSdhcSrcExt }`
 SDHC clock source.
- enum `clock_sai_src_k64f12_t`{
 `kClockSaiSrcSysClk = 0U,`
 `kClockSaiSrcOsc0erClk = 1U,`
 `kClockSaiSrcPllClk = 3U }`
 SAI clock source.
- enum `clock_pllfl_sel_k64f12_t`{
 `kClockPllFllSelFll = 0U,`
 `kClockPllFllSelPll = 1U,`
 `kClockPllFllSelIrc48M = 3U }`
 SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k64f12_t`{
 `kClockEr32kSrcOsc0 = 0U,`
 `kClockEr32kSrcRtc = 2U,`
 `kClockEr32kSrcLpo = 3U }`
 SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k64f12_t`{
 `kClockClkoutSelFlexbusClk = 0U,`
 `kClockClkoutSelFlashClk = 2U,`
 `kClockClkoutSelLpoClk = 3U,`
 `kClockClkoutSelMcgIrClk = 4U,`
 `kClockClkoutSelRtc = 5U,`
 `kClockClkoutSelOsc0erClk = 6U,`
 `kClockClkoutSelIrc48M = 7U }`
 SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k64f12_t`{
 `kClockRtcoutSrc1Hz,`
 `kClockRtcoutSrc32kHz }`
 SIM RTCCLKOUTSEL clock source select.
- enum `sim_usbsstby_mode_k64f12_t`{
 `kSimUsbstbyNoRegulator,`
 `kSimUsbstbyWithRegulator }`
 SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k64f12_t`{

```
kSimUsbstbyNoRegulator,
kSimUsbstbyWithRegulator }
```

SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.

- enum `sim_adc_pretrg_sel_k64f12_t` {

 `kSimAdcPretrgselA,`

 `kSimAdcPretrgselB }`

SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_k64f12_t` {

 `kSimAdcTrgselExt = 0U,`

 `kSimAdcTrgSelHighSpeedComp0 = 1U,`

 `kSimAdcTrgSelHighSpeedComp1 = 2U,`

 `kSimAdcTrgSelHighSpeedComp2 = 3U,`

 `kSimAdcTrgSelPit0 = 4U,`

 `kSimAdcTrgSelPit1 = 5U,`

 `kSimAdcTrgSelPit2 = 6U,`

 `kSimAdcTrgSelPit3 = 7U,`

 `kSimAdcTrgSelFtm0 = 8U,`

 `kSimAdcTrgSelFtm1 = 9U,`

 `kSimAdcTrgSelFtm2 = 10U,`

 `kSimAdcTrgSelFtm3 = 11U,`

 `kSimAdcTrgSelRtcAlarm = 12U,`

 `kSimAdcTrgSelRtcSec = 13U,`

 `kSimAdcTrgSelLptimer = 14U }`

SIM ADCx trigger select.

- enum `sim_uart_rxsrc_k64f12_t` {

 `kSimUartRxsrcPin,`

 `kSimUartRxsrcCmp0,`

 `kSimUartRxsrcCmp1 }`

SIM receive data source select.
- enum `sim_uart_txsrc_k64f12_t` {

 `kSimUartTxsrcPin,`

 `kSimUartTxsrcFtm1,`

 `kSimUartTxsrcFtm2 }`

SIM transmit data source select.

- enum `sim_ftm_trg_src_k64f12_t` {

 `kSimFtmTrgSrc0,`

 `kSimFtmTrgSrc1 }`

SIM FlexTimer x trigger y select.
- enum `sim_ftm_clk_sel_k64f12_t` {

 `kSimFtmClkSel0,`

 `kSimFtmClkSel1 }`

SIM FlexTimer external clock select.
- enum `sim_ftm_ch_src_k64f12_t` {

 `kSimFtmChSrc0,`

 `kSimFtmChSrc1,`

 `kSimFtmChSrc2,`

SIM HAL driver

- ```
kSimFtmChSrc3 }
 SIM FlexTimer x channel y input capture source select.
• enum sim_ftm_ftl_sel_k64f12_t {
 kSimFtmFltSel0,
 kSimFtmFltSel1 }
 SIM FlexTimer x Fault y select.
• enum sim_tpm_clk_sel_k64f12_t {
 kSimTpmClkSel0,
 kSimTpmClkSel1 }
 SIM Timer/PWM external clock select.
• enum sim_tpm_ch_src_k64f12_t {
 kSimTpmChSrc0,
 kSimTpmChSrc1,
 kSimTpmChSrc2,
 kSimTpmChSrc3 }
 SIM Timer/PWM x channel y input capture source select.
• enum sim_cmtuartpad_strenght_k64f12_t {
 kSimCmtuartSinglePad,
 kSimCmtuartDualPad }
 SIM CMT/UART pad drive strength.
• enum sim_ptd7pad_strenght_k64f12_t {
 kSimPtd7padSinglePad,
 kSimPtd7padDualPad }
 SIM PTD7 pad drive strength.
• enum sim_flexbus_security_level_k64f12_t {
 kSimFbslLevel0,
 kSimFbslLevel1,
 kSimFbslLevel2,
 kSimFbslLevel3 }
 SIM FlexBus security level.
• enum sim_clock_gate_name_k64f12_t
 Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.
```

### 40.2.21.2 Macro Definition Documentation

#### 40.2.21.2.1 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

### 40.2.21.3 Enumeration Type Documentation

#### 40.2.21.3.1 enum clock\_wdog\_src\_k64f12\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.

**40.2.21.3.2 enum clock\_trace\_src\_k64f12\_t**

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

**40.2.21.3.3 enum clock\_port\_filter\_src\_k64f12\_t**

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

**40.2.21.3.4 enum clock\_lptmr\_src\_k64f12\_t**

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

**40.2.21.3.5 enum clock\_time\_src\_k64f12\_t**

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.  
*kClockTimeSrcPlifllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTimeSrcOsc0erClk* OSCERCLK clock.  
*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

**40.2.21.3.6 enum clock\_rmii\_src\_k64f12\_t**

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.  
*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

**40.2.21.3.7 enum clock\_usbfs\_src\_k64f12\_t**

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPlifllSel* Clock divider USB FS clock.

## SIM HAL driver

### 40.2.21.3.8 enum clock\_flexcan\_src\_k64f12\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.

*kClockFlexcanSrcBusClk* Bus clock.

### 40.2.21.3.9 enum clock\_sdhc\_src\_k64f12\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.

*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].

*kClockSdhcSrcOsc0erClk* OSCERCLK clock.

*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 40.2.21.3.10 enum clock\_sai\_src\_k64f12\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.

*kClockSaiSrcOsc0erClk* OSC0ERCLK.

*kClockSaiSrcPllClk* MCGPLLCLK.

### 40.2.21.3.11 enum clock\_pllfll\_sel\_k64f12\_t

Enumerator

*kClockPllFllSelFll* Fll clock.

*kClockPllFllSelPll* Pll0 clock.

*kClockPllFllSelIrc48M* IRC48MCLK.

### 40.2.21.3.12 enum clock\_er32k\_src\_k64f12\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).

*kClockEr32kSrcRtc* RTC 32k clock .

*kClockEr32kSrcLpo* LPO clock.

**40.2.21.3.13 enum clock\_clkout\_src\_k64f12\_t**

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

**40.2.21.3.14 enum clock\_rtcout\_src\_k64f12\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

**40.2.21.3.15 enum sim\_usbsstby\_mode\_k64f12\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes  
*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

**40.2.21.3.16 enum sim\_usbvstby\_mode\_k64f12\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes  
*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

**40.2.21.3.17 enum sim\_adc\_pretrg\_sel\_k64f12\_t**

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

## SIM HAL driver

### 40.2.21.3.18 enum sim\_adc\_trg\_sel\_k64f12\_t

Enumerator

- kSimAdcTrgselExt* External trigger.
- kSimAdcTrgSelHighSpeedComp0* High speed comparator 0 output.
- kSimAdcTrgSelHighSpeedComp1* High speed comparator 1 output.
- kSimAdcTrgSelHighSpeedComp2* High speed comparator 2 output.
- kSimAdcTrgSelPit0* PIT trigger 0.
- kSimAdcTrgSelPit1* PIT trigger 1.
- kSimAdcTrgSelPit2* PIT trigger 2.
- kSimAdcTrgSelPit3* PIT trigger 3.
- kSimAdcTrgSelFtm0* FTM0 trigger.
- kSimAdcTrgSelFtm1* FTM1 trigger.
- kSimAdcTrgSelFtm2* FTM2 trigger.
- kSimAdcTrgSelFtm3* FTM3 trigger.
- kSimAdcTrgSelRtcAlarm* RTC alarm.
- kSimAdcTrgSelRtcSec* RTC seconds.
- kSimAdcTrgSelLptimer* Low-power timer trigger.

### 40.2.21.3.19 enum sim\_uart\_rxsrc\_k64f12\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

### 40.2.21.3.20 enum sim\_uart\_txsrc\_k64f12\_t

Enumerator

- kSimUartTxsrcPin* UARTx\_TX Pin.
- kSimUartTxsrcFtm1* UARTx\_TX pin modulated with FTM1 channel 0 output.
- kSimUartTxsrcFtm2* UARTx\_TX pin modulated with FTM2 channel 0 output.

### 40.2.21.3.21 enum sim\_ftm\_trg\_src\_k64f12\_t

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

**40.2.21.3.22 enum sim\_ftm\_clk\_sel\_k64f12\_t**

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.*kSimFtmClkSel1* FTM CLKIN1 pin.**40.2.21.3.23 enum sim\_ftm\_ch\_src\_k64f12\_t**

Enumerator

*kSimFtmChSrc0* FlexTimer x channel y input capture source 0.*kSimFtmChSrc1* FlexTimer x channel y input capture source 1.*kSimFtmChSrc2* FlexTimer x channel y input capture source 2.*kSimFtmChSrc3* FlexTimer x channel y input capture source 3.**40.2.21.3.24 enum sim\_ftm\_flt\_sel\_k64f12\_t**

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.*kSimFtmFltSel1* FlexTimer x fault y select 1.**40.2.21.3.25 enum sim\_tpm\_clk\_sel\_k64f12\_t**

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.**40.2.21.3.26 enum sim\_tpm\_ch\_src\_k64f12\_t**

Enumerator

*kSimTpmChSrc0* TPM x channel y input capture source 0.*kSimTpmChSrc1* TPM x channel y input capture source 1.*kSimTpmChSrc2* TPM x channel y input capture source 2.*kSimTpmChSrc3* TPM x channel y input capture source 3.**40.2.21.3.27 enum sim\_cmtuartpad\_strenght\_k64f12\_t**

Enumerator

*kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.*kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

## SIM HAL driver

### 40.2.21.3.28 enum sim\_ptd7pad\_strenght\_k64f12\_t

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.

*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 40.2.21.3.29 enum sim\_flexbus\_security\_level\_k64f12\_t

Enumerator

*kSimFbslLevel0* FlexBus security level 0.

*kSimFbslLevel1* FlexBus security level 1.

*kSimFbslLevel2* FlexBus security level 2.

*kSimFbslLevel3* FlexBus security level 3.

### 40.2.21.3.30 enum sim\_clock\_gate\_name\_k64f12\_t

# Chapter 41

## System Mode Controller (SMC)

### 41.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the System Mode Controller (SMC) block of Kinetis devices.

### Modules

- SMC HAL driver

### 41.2 SMC HAL driver

#### 41.2.1 Overview

The section describes the programming interface of the SMC HAL driver. The System Mode Controller (SMC) sequences the system in and out of all low-power stop and run modes. Specifically, it monitors events to trigger transitions between the power modes while controlling the power, clocks, and memories of the system to achieve the power consumption and functionality of that mode. It also provides a set of functions to configure the power mode protection, the power mode, and other configuration settings.

#### 41.2.2 Power Mode Configuration APIs

- Power Mode Protection configurations APIs Allow the Very Low Power mode / Read the current mode
- Allow the Low Leakage Stop mode / Read the current mode
- Allow the Very Low Leakage Stop mode / Read the current mode
- Power Mode Control configurations APIs
- VLLS Mode Control configurations APIs
- Power Mode Status APIs

This is an example of the SMC manager APIs.

```
#include "fsl_smcc_hal.h"

// power mode config structure //
smc_power_mode_config_t smcConfig;

// power mode and option mode //
smcConfig.lpwuiOption = false;
smcConfig.porOption = false;
smcConfig.powerModeName = kPowerModeRun;

// set the power mode //
SMC_HAL_SetMode(&smcConfig);
```

#### Control register access APIs

- Power mode configurations register access
- VLLS mode configurations register access
- Power Mode Status access

This is an example of the SMC HAL access APIs.

```
#include "fsl_smcc_hal.h"

// protection mode //
smc_power_mode_protection_config_t smcProtConfig;

// set to allow entering specific protect mode //
smcProtConfig.llsProt = true;
smcProtConfig.vllsProt = true;
smcProtConfig.vlpProt = true;
```

```
// set protection modes //
SMC_HAL_SetProtection(&smcProtConfig);
```

## Files

- file [fsl\\_smc\\_hal.h](#)

## Data Structures

- struct [smc\\_power\\_mode\\_protection\\_config\\_t](#)  
*Power mode protection configuration.* [More...](#)
- struct [smc\\_power\\_mode\\_config\\_t](#)  
*Power mode control configuration used for calling the SMC\_SYS\_SetPowerMode API.* [More...](#)

## Enumerations

- enum [power\\_modes\\_t](#)  
*Power Modes.*
- enum [smc\\_hal\\_error\\_code\\_t](#) {  
 kSmcHalSuccess,  
 kSmcHalNoSuchModeName,  
 kSmcHalAlreadyInTheState,  
 kSmcHalFailed }  
*Error code definition for the system mode controller manager APIs.*
- enum [power\\_mode\\_stat\\_t](#) {  
 kStatRun = 0x01,  
 kStatStop = 0x02,  
 kStatVlpr = 0x04,  
 kStatVlpw = 0x08,  
 kStatVlps = 0x10,  
 kStatVlls = 0x40 }  
*Power Modes in PMSTAT.*
- enum [power\\_modes\\_protect\\_t](#) {  
 kAllowHsrun,  
 kAllowVlp,  
 kAllowLls,  
 kAllowVlls }  
*Power Modes Protection.*
- enum [smc\\_run\\_mode\\_t](#) {  
 kSmcRun ,  
 kSmcVlpr }  
*Run mode definition.*
- enum [smc\\_stop\\_mode\\_t](#) {

## SMC HAL driver

- ```
kSmcStop = 0U,  
kSmcReservedStop1 = 1U,  
kSmcVlps = 2U,  
kSmcVlls = 4U }  
    Stop mode definition.  
• enum smc_stop_submode_t  
    VLLS/LLS stop sub mode definition.  
• enum smc_lpwui_option_t {  
    kSmcLpwuiEnabled,  
    kSmcLpwuiDisabled }  
    Low Power Wake Up on Interrupt option.  
• enum smc_pstop_option_t {  
    kSmcPstopStop,  
    kSmcPstopStop1,  
    kSmcPstopStop2 }  
    Partial STOP option.  
• enum smc_por_option_t {  
    kSmcPorEnabled,  
    kSmcPorDisabled }  
    POR option.  
• enum smc_ram2_option_t {  
    kSmcRam2DisPowered,  
    kSmcRam2Powered }  
    RAM2 power option.  
• enum smc_lpo_option_t {  
    kSmcLpoEnabled,  
    kSmcLpoDisabled }  
    LPO power option.  
• enum smc_power_options_t {  
    kSmcOptionLpwui,  
    kSmcOptionPropo }  
    Power mode control options.
```

System mode controller APIs

- `smc_hal_error_code_t SMC_HAL_SetMode (uint32_t baseAddr, const smc_power_mode_config_t *powerModeConfig)`
Configures the power mode.
- `void SMC_HAL_SetProtection (uint32_t baseAddr, smc_power_mode_protection_config_t *protectConfig)`
Configures all power mode protection settings.
- `void SMC_HAL_SetProtectionMode (uint32_t baseAddr, power_modes_protect_t protect, bool allow)`
Configures the individual power mode protection settings.
- `bool SMC_HAL_GetProtectionMode (uint32_t baseAddr, power_modes_protect_t protect)`
Gets the current power mode protection setting.
- `void SMC_HAL_SetRunMode (uint32_t baseAddr, smc_run_mode_t runMode)`

- Configures the the RUN mode control setting.
- **smc_run_mode_t SMC_HAL_GetRunMode** (uint32_t baseAddr)
Gets the current RUN mode configuration setting.
- void **SMC_HAL_SetStopMode** (uint32_t baseAddr, **smc_stop_mode_t** stopMode)
Configures the STOP mode control setting.
- **smc_stop_mode_t SMC_HAL_GetStopMode** (uint32_t baseAddr)
Gets the current STOP mode control settings.
- void **SMC_HAL_SetStopSubMode** (uint32_t baseAddr, **smc_stop_submode_t** stopSubMode)
Configures the stop sub mode control setting.
- **smc_stop_submode_t SMC_HAL_GetStopSubMode** (uint32_t baseAddr)
Gets the current stop submode configuration settings.
- **power_mode_stat_t SMC_HAL_GetStat** (uint32_t baseAddr)
Gets the current power mode stat.

41.2.3 Data Structure Documentation

41.2.3.1 struct smc_power_mode_protection_config_t

Data Fields

- bool **vlpProt**
VLP protect.
- bool **vllsProt**
VLLS protect.

41.2.3.2 struct smc_power_mode_config_t

Data Fields

- **power_modes_t powerModeName**
Power mode(enum), see power_modes_t.
- **smc_stop_submode_t stopSubMode**
Stop submode(enum), see smc_stop_submode_t.

41.2.4 Enumeration Type Documentation

41.2.4.1 enum smc_hal_error_code_t

Enumerator

kSmcHalSuccess Success.

kSmcHalNoSuchModeName Cannot find the mode name specified.

kSmcHalAlreadyInTheState Already in the required state.

kSmcHalFailed Unknown error, operation failed.

SMC HAL driver

41.2.4.2 enum power_mode_stat_t

Enumerator

kStatRun 0000_0001 - Current power mode is RUN
kStatStop 0000_0010 - Current power mode is STOP
kStatVlpr 0000_0100 - Current power mode is VLPR
kStatVlpw 0000_1000 - Current power mode is VLPW
kStatVlps 0001_0000 - Current power mode is VLPS
kStatVlls 0100_0000 - Current power mode is VLLS

41.2.4.3 enum power_modes_protect_t

Enumerator

kAllowHsrun Allow High Speed Run mode.
kAllowVlp Allow Very-Low-Power Modes.
kAllowLls Allow Low-Leakage Stop Mode.
kAllowVlls Allow Very-Low-Leakage Stop Mode.

41.2.4.4 enum smc_run_mode_t

Enumerator

kSmcRun normal RUN mode
kSmcVlpr Very-Low-Power RUN mode.

41.2.4.5 enum smc_stop_mode_t

Enumerator

kSmcStop Normal STOP mode.
kSmcReservedStop1 Reserved.
kSmcVlps Very-Low-Power STOP mode.
kSmcVlls Very-Low-Leakage Stop mode.

41.2.4.6 enum smc_lpwui_option_t

Enumerator

kSmcLpwuiEnabled Low Power Wake Up on Interrupt enabled.
kSmcLpwuiDisabled Low Power Wake Up on Interrupt disabled.

41.2.4.7 enum smc_pstop_option_t

Enumerator

kSmcPstopStop STOP - Normal Stop mode.

kSmcPstopStop1 Partial Stop with both system and bus clocks disabled.

kSmcPstopStop2 Partial Stop with system clock disabled and bus clock enabled.

41.2.4.8 enum smc_por_option_t

Enumerator

kSmcPorEnabled POR detect circuit is enabled in VLLS0.

kSmcPorDisabled POR detect circuit is disabled in VLLS0.

41.2.4.9 enum smc_ram2_option_t

Enumerator

kSmcRam2DisPowered RAM2 not powered in LLS2/VLLS2.

kSmcRam2Powered RAM2 powered in LLS2/VLLS2.

41.2.4.10 enum smc_lpo_option_t

Enumerator

kSmcLpoEnabled LPO clock is enabled in LLS/VLLSx.

kSmcLpoDisabled LPO clock is disabled in LLS/VLLSx.

41.2.4.11 enum smc_power_options_t

Enumerator

kSmcOptionLpwui Low Power Wake Up on Interrupt.

kSmcOptionPropo POR option.

41.2.5 Function Documentation

41.2.5.1 smc_hal_error_code_t SMC_HAL_SetMode (uint32_t baseAddr, const smc_power_mode_config_t * powerModeConfig)

This function configures the power mode control for both run, stop, and stop sub mode if needed. Also it configures the power options for a specific power mode. An application should follow the proper procedure

SMC HAL driver

to configure and switch power modes between different run and stop modes. For proper procedures and supported power modes, see an appropriate chip reference manual. See the [smc_power_mode_config_t](#) for required parameters to configure the power mode and the supported options. Other options may need to be individually configured through the HAL driver. See the HAL driver header file for details.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>powerModeConfig</i>	Power mode configuration structure smc_power_mode_config_t

Returns

errorCode SMC error code

41.2.5.2 void SMC_HAL_SetProtection (uint32_t *baseAddr*, smc_power_mode_protection_config_t * *protectConfig*)

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_config_t](#). An application should provide the protect settings for all supported power modes on the chip. 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. If the user has only a single option to set, either use this function or use the individual set function.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>protectConfig</i>	Configurations for the supported power mode protect settings <ul style="list-style-type: none">• See smc_power_mode_protection_config_t for details.

41.2.5.3 void SMC_HAL_SetProtectionMode (uint32_t *baseAddr*, power_modes_protect_t *protect*, bool *allow*)

This function only configures the power mode protection settings for a specified power mode on the specified chip family. The available power modes are defined in the [smc_power_mode_protection_config_t](#). See the reference manual for details. This register can only write once after the power reset.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>protect</i>	Power mode to set for protection
<i>allow</i>	Allow or not allow the power mode protection

41.2.5.4 **bool SMC_HAL_GetProtectionMode (uint32_t *baseAddr*, power_modes_protect_t *protect*)**

This function gets the current power mode protection settings for a specified power mode.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>protect</i>	Power mode to set for protection

Returns

state Status of the protection setting

- true: Allowed
- false: Not allowed

41.2.5.5 **void SMC_HAL_SetRunMode (uint32_t *baseAddr*, smc_run_mode_t *runMode*)**

This function sets the run mode settings, for example, normal run mode, very lower power run mode, etc. See the *smc_run_mode_t* for supported run mode on the chip family and the reference manual for details about the run mode.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>runMode</i>	Run mode setting defined in <i>smc_run_mode_t</i>

41.2.5.6 **smc_run_mode_t SMC_HAL_GetRunMode (uint32_t *baseAddr*)**

This function gets the run mode settings. See the *smc_run_mode_t* for a supported run mode on the chip family and the reference manual for details about the run mode.

SMC HAL driver

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
-----------------	--

Returns

setting Run mode configuration setting

41.2.5.7 void SMC_HAL_SetStopMode (uint32_t *baseAddr*, smc_stop_mode_t *stopMode*)

This function sets the stop mode settings, for example, normal stop mode, very lower power stop mode, etc. See the smc_stop_mode_t for supported stop mode on the chip family and the reference manual for details about the stop mode.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>stopMode</i>	Stop mode defined in smc_stop_mode_t

41.2.5.8 smc_stop_mode_t SMC_HAL_GetStopMode (uint32_t *baseAddr*)

This function gets the stop mode settings, for example, normal stop mode, very lower power stop mode, etc. See the smc_stop_mode_t for supported stop mode on the chip family and the reference manual for details about the stop mode.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
-----------------	--

Returns

setting Current stop mode configuration setting

41.2.5.9 void SMC_HAL_SetStopSubMode (uint32_t *baseAddr*, smc_stop_submode_t *stopSubMode*)

This function sets the stop submode settings. Some of the stop mode further supports submodes. See the smc_stop_submode_t for supported stop submodes and the reference manual for details about the submodes for a specific stop mode.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
<i>stopSubMode</i>	Stop submode setting defined in smc_stop_submode_t

41.2.5.10 smc_stop_submode_t SMC_HAL_GetStopSubMode (uint32_t *baseAddr*)

This function gets the stop submode settings. Some of the stop mode further support submodes. See the smc_stop_submode_t for supported stop submodes and the reference manual for details about the submode for a specific stop mode.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
-----------------	--

Returns

setting Current stop submode setting

41.2.5.11 power_mode_stat_t SMC_HAL_GetStat (uint32_t *baseAddr*)

This function returns the current power mode stat. Once application switches the power mode, it should always check the stat to check whether it runs into the specified mode or not. An 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 _power_mode_stat for information about the power stat.

Parameters

<i>baseAddr</i>	Base address for current SMC instance.
-----------------	--

Returns

stat Current power mode stat

Chapter 42

Clock Manager (Clock)

The Kinetis SDK Clock Manager provides a set of API/services to configure the clock-related IPs, such as MCG, SIM, etc.

42.1 Overview

Modules

- [K02F12810](#)
The data structure definition for K02F12810 clock manager.
- [K22F12810](#)
The data structure definition for K22F12810 clock manager.
- [K22F25612](#)
The data structure definition for K22F25612 clock manager.
- [K22F51212](#)
The data structure definition for K22F51212 clock manager.
- [K24F12](#)
The data structure definition for K24F12 clock manager.
- [K24F25612](#)
The data structure definition for K24F25612 clock manager.
- [K60D10](#)
The data structure definition for K60D10 clock manager.
- [K64F12](#)
The data structure definition for K64F12 clock manager.
- [KL03Z4](#)
The data structure definition for KL03Z4 clock manager.
- [KL46Z4](#)
The data structure definition for KL46Z4 clock manager.
- [KV10Z7](#)
The data structure definition for KV10Z7 clock manager.
- [KV30F12810](#)
The data structure definition for KV30F12810 clock manager.
- [KV31F12810](#)
The data structure definition for KV31F12810 clock manager.
- [KV31F25612](#)
The data structure definition for KV31F25612 clock manager.
- [KV31F51212](#)
The data structure definition for KV31F51212 clock manager.

Files

- file [fsl_clock_manager.h](#)
- file [fsl_clock_MK63F12.h](#)
- file [fsl_clock_MKL25Z4.h](#)

Overview

Data Structures

- struct [mcg_config_t](#)
MCG configure structure for mode change. [More...](#)
- struct [clock_manager_user_config_t](#)
Clock configuration structure. [More...](#)
- struct [clock_notify_struct_t](#)
Clock notification structure passed to clock callback function. [More...](#)
- struct [clock_manager_callback_user_config_t](#)
Structure for callback function and its parameter. [More...](#)
- struct [clock_manager_state_t](#)
Clock manager state structure. [More...](#)

Macros

- #define [CPU_LPO_CLK_HZ](#) 1000U
Frequency of LPO.

Typedefs

- typedef [clock_manager_error_code_t](#)(* [clock_manager_callback_t](#))([clock_notify_struct_t](#) *notify, void *callbackData)
Type of clock callback functions.

Enumerations

- enum [clock_names_t](#) {
 [kCoreClock](#),
 [kSystemClock](#),
 [kPlatformClock](#),
 [kBusClock](#),
 [kFlexBusClock](#),
 [kFlashClock](#),
 [kOsc32kClock](#),
 [kOsc0ErClock](#),
 [kOsc1ErClock](#),
 [kOsc0ErClockUndiv](#),
 [kIrc48mClock](#),
 [kRtcoutClock](#),
 [kMcgFfClock](#),
 [kMcgFllClock](#),
 [kMcgPll0Clock](#),
 [kMcgPll1Clock](#),
 [kMcgOutClock](#),
 [kMcgIrClock](#),
 [kLpoClock](#) }

Clock name used to get clock frequency.

- enum `clock_manager_error_code_t` {

 `kClockManagerSuccess`,

 `kClockManagerError`,

 `kClockManagerNoSuchClockName`,

 `kClockManagerInvalidParam`,

 `kClockManagerErrorOutOfRange`,

 `kClockManagerErrorNotificationBefore`,

 `kClockManagerErrorNotificationAfter`,

 `kClockManagerErrorUnknown` }

Error code definition for the clock manager APIs.

- enum `clock_manager_notify_t` {

 `kClockManagerNotifyRecover` = 0x00U,

 `kClockManagerNotifyBefore` = 0x01U,

 `kClockManagerNotifyAfter` = 0x02U }

The clock notification type.

- enum `clock_manager_callback_type_t` {

 `kClockManagerCallbackBefore` = 0x01U,

 `kClockManagerCallbackAfter` = 0x02U,

 `kClockManagerCallbackBeforeAfter` = 0x03U }

The callback type, indicates what kinds of notification this callback handles.

- enum `clock_manager_policy_t` {

 `kClockManagerPolicyAgreement`,

 `kClockManagerPolicyForcible` }

Clock transition policy.

Functions

- `clock_manager_error_code_t CLOCK_SYS_GetFreq (clock_names_t clockName, uint32_t *frequency)`

Gets the clock frequency for a specific clock name.
- `uint32_t CLOCK_SYS_GetCoreClockFreq (void)`

Get core clock frequency.
- `uint32_t CLOCK_SYS_GetSystemClockFreq (void)`

Get system clock frequency.
- `uint32_t CLOCK_SYS_GetBusClockFreq (void)`

Get bus clock frequency.
- `uint32_t CLOCK_SYS_GetFlashClockFreq (void)`

Get flash clock frequency.
- `static uint32_t CLOCK_SYS_GetLpoClockFreq (void)`

Get LPO clock frequency.
- `static void CLOCK_SYS_SetOutDiv1 (uint8_t outdiv1)`

Sets the clock out divider1 setting(OUTDIV1).
- `static uint8_t CLOCK_SYS_GetOutDiv1 (void)`

Gets the clock out divider1 setting(OUTDIV1).
- `static void CLOCK_SYS_SetOutDiv2 (uint8_t outdiv2)`

Sets the clock out divider2 setting(OUTDIV2).
- `static uint8_t CLOCK_SYS_GetOutDiv2 (void)`

Gets the clock out divider2 setting(OUTDIV2).
- `static void CLOCK_SYS_SetOutDiv4 (uint8_t outdiv4)`

Overview

- static uint8_t **CLOCK_SYS_GetOutDiv4** (void)
Gets the clock out divider4 setting(OUTDIV4).
- static void **CLOCK_SYS_SetOutDiv** (uint8_t outdiv1, uint8_t outdiv2, uint8_t outdiv3, uint8_t outdiv4)
Sets the clock out divider4 setting(OUTDIV4).
- static void **CLOCK_SYS_SetOutDiv** (uint8_t *outdiv1, uint8_t *outdiv2, uint8_t *outdiv3, uint8_t *outdiv4)
Sets the clock out dividers setting.
- static void **CLOCK_SYS_SetOutDiv** (uint8_t *outdiv1, uint8_t *outdiv2, uint8_t *outdiv3, uint8_t *outdiv4)
Gets the clock out dividers setting.
- uint32_t **CLOCK_SYS_GetPllfllClockFreq** (void)
Get the MCGPLLCLK/MCGFLLCLK/IRC48MCLK clock frequency.
- static void **CLOCK_SYS_SetPllfllSel** (clock_pllfl_sel_t setting)
Set PLL/FLL clock selection.
- static clock_pllfl_sel_t **CLOCK_SYS_GetPllfllSel** (void)
Get PLL/FLL clock selection.
- static uint32_t **CLOCK_SYS_GetFixedFreqClockFreq** (void)
Gets the MCGFFCLK clock frequency.
- static uint32_t **CLOCK_SYS_GetInternalRefClockFreq** (void)
Get internal reference clock frequency.
- uint32_t **CLOCK_SYS_GetExternalRefClock32kFreq** (void)
Gets the external reference 32k clock frequency.
- static void **CLOCK_SYS_SetExternalRefClock32kSrc** (clock_er32k_src_t src)
Set the clock selection of ERCLK32K.
- static clock_er32k_src_t **CLOCK_SYS_GetExternalRefClock32kSrc** (void)
Get the clock selection of ERCLK32K.
- uint32_t **CLOCK_SYS_GetOsc0ExternalRefClockFreq** (void)
Gets the OSC0ERCLK frequency.
- uint32_t **CLOCK_SYS_GetOsc0ExternalRefClockUndivFreq** (void)
Gets the OSC0ERCLK_UNDIV frequency.
- uint32_t **CLOCK_SYS_GetWdogFreq** (uint32_t instance, clock_wdog_src_t wdogSrc)
Gets the watch dog clock frequency.
- static clock_trace_src_t **CLOCK_SYS_GetTraceSrc** (uint32_t instance)
Gets the debug trace clock source.
- static void **CLOCK_SYS_SetTraceSrc** (uint32_t instance, clock_trace_src_t src)
Sets the debug trace clock source.
- uint32_t **CLOCK_SYS_GetTraceFreq** (uint32_t instance)
Gets the debug trace clock frequency.
- uint32_t **CLOCK_SYS_GetPortFilterFreq** (uint32_t instance, clock_port_filter_src_t src)
Gets PORTx digital input filter clock frequency.
- uint32_t **CLOCK_SYS_GetLptmrFreq** (uint32_t instance, clock_lptmr_src_t lptmrSrc)
Gets LPTMRx pre-scaler/glitch filter clock frequency.
- static uint32_t **CLOCK_SYS_GetEwmFreq** (uint32_t instance)
Gets the clock frequency for EWM module.
- static uint32_t **CLOCK_SYS_GetFtfFreq** (uint32_t instance)
Gets the clock frequency for FTF module.
- static uint32_t **CLOCK_SYS_GetCrcFreq** (uint32_t instance)
Gets the clock frequency for CRC module.
- static uint32_t **CLOCK_SYS_GetCmpFreq** (uint32_t instance)
Gets the clock frequency for CMP module.
- static uint32_t **CLOCK_SYS_GetVrefFreq** (uint32_t instance)
Gets the clock frequency for VREF module.

- static uint32_t **CLOCK_SYS_GetPdbFreq** (uint32_t instance)

Gets the clock frequency for PDB module.
- static uint32_t **CLOCK_SYS_GetPitFreq** (uint32_t instance)

Gets the clock frequency for PIT module.
- static uint32_t **CLOCK_SYS_GetSpiFreq** (uint32_t instance)

Gets the clock frequency for SPI module.
- static uint32_t **CLOCK_SYS_GetI2cFreq** (uint32_t instance)

Gets the clock frequency for I2C module.
- static uint32_t **CLOCK_SYS_GetAdcAltFreq** (uint32_t instance)

Gets ADC alternate clock frequency.
- static uint32_t **CLOCK_SYS_GetAdcAlt2Freq** (uint32_t instance)

Gets ADC alternate 2 clock frequency.
- static uint32_t **CLOCK_SYS_GetFtmFixedFreq** (uint32_t instance)

Gets FTM fixed frequency clock frequency.
- uint32_t **CLOCK_SYS_GetFtmExternalFreq** (uint32_t instance)

Gets FTM external clock frequency.
- static **sim_ftm_clk_sel_t CLOCK_SYS_GetFtmExternalSrc** (uint32_t instance)

Gets FTM external clock source.
- static void **CLOCK_SYS_SetFtmExternalSrc** (uint32_t instance, **sim_ftm_clk_sel_t** ftmSrc)

Sets FTM external clock source.
- uint32_t **CLOCK_SYS_GetUartFreq** (uint32_t instance)

Gets the clock frequency for UART module.
- static uint32_t **CLOCK_SYS_GetGpioFreq** (uint32_t instance)

Gets the clock frequency for GPIO module.
- static void **CLOCK_SYS_EnableDmaClock** (uint32_t instance)

Enable the clock for DMA module.
- static void **CLOCK_SYS_DisableDmaClock** (uint32_t instance)

Disable the clock for DMA module.
- static bool **CLOCK_SYS_GetDmaGateCmd** (uint32_t instance)

Get the the clock gate state for DMA module.
- static void **CLOCK_SYS_EnableDmamuxClock** (uint32_t instance)

Enable the clock for DMAMUX module.
- static void **CLOCK_SYS_DisableDmamuxClock** (uint32_t instance)

Disable the clock for DMAMUX module.
- static bool **CLOCK_SYS_GetDmamuxGateCmd** (uint32_t instance)

Get the the clock gate state for DMAMUX module.
- void **CLOCK_SYS_EnablePortClock** (uint32_t instance)

Enable the clock for PORT module.
- void **CLOCK_SYS_DisablePortClock** (uint32_t instance)

Disable the clock for PORT module.
- bool **CLOCK_SYS_GetPortGateCmd** (uint32_t instance)

Get the the clock gate state for PORT module.
- static void **CLOCK_SYS_EnableEwmClock** (uint32_t instance)

Enable the clock for EWM module.
- static void **CLOCK_SYS_DisableEwmClock** (uint32_t instance)

Disable the clock for EWM module.
- static bool **CLOCK_SYS_GetEwmGateCmd** (uint32_t instance)

Get the the clock gate state for EWM module.
- static void **CLOCK_SYS_EnableFtfClock** (uint32_t instance)

Enable the clock for FTF module.
- static void **CLOCK_SYS_DisableFtfClock** (uint32_t instance)

Overview

Disable the clock for FTF module.

- static bool **CLOCK_SYS_GetFtfGateCmd** (uint32_t instance)
Get the the clock gate state for FTF module.
- static void **CLOCK_SYS_EnableCrcClock** (uint32_t instance)
Enable the clock for CRC module.
- static void **CLOCK_SYS_DisableCrcClock** (uint32_t instance)
Disable the clock for CRC module.
- static bool **CLOCK_SYS_GetCrcGateCmd** (uint32_t instance)
Get the the clock gate state for CRC module.
- void **CLOCK_SYS_EnableAdcClock** (uint32_t instance)
Enable the clock for ADC module.
- void **CLOCK_SYS_DisableAdcClock** (uint32_t instance)
Disable the clock for ADC module.
- bool **CLOCK_SYS_GetAdcGateCmd** (uint32_t instance)
Get the the clock gate state for ADC module.
- static void **CLOCK_SYS_EnableCmpClock** (uint32_t instance)
Enable the clock for CMP module.
- static void **CLOCK_SYS_DisableCmpClock** (uint32_t instance)
Disable the clock for CMP module.
- static bool **CLOCK_SYS_GetCmpGateCmd** (uint32_t instance)
Get the the clock gate state for CMP module.
- void **CLOCK_SYS_EnableDacClock** (uint32_t instance)
Enable the clock for DAC module.
- void **CLOCK_SYS_DisableDacClock** (uint32_t instance)
Disable the clock for DAC module.
- bool **CLOCK_SYS_GetDacGateCmd** (uint32_t instance)
Get the the clock gate state for DAC module.
- static void **CLOCK_SYS_EnableVrefClock** (uint32_t instance)
Enable the clock for VREF module.
- static void **CLOCK_SYS_DisableVrefClock** (uint32_t instance)
Disable the clock for VREF module.
- static bool **CLOCK_SYS_GetVrefGateCmd** (uint32_t instance)
Get the the clock gate state for VREF module.
- static void **CLOCK_SYS_EnablePdbClock** (uint32_t instance)
Enable the clock for PDB module.
- static void **CLOCK_SYS_DisablePdbClock** (uint32_t instance)
Disable the clock for PDB module.
- static bool **CLOCK_SYS_GetPdbGateCmd** (uint32_t instance)
Get the the clock gate state for PDB module.
- void **CLOCK_SYS_EnableFtmClock** (uint32_t instance)
Enable the clock for FTM module.
- void **CLOCK_SYS_DisableFtmClock** (uint32_t instance)
Disable the clock for FTM module.
- bool **CLOCK_SYS_GetFtmGateCmd** (uint32_t instance)
Get the the clock gate state for FTM module.
- static void **CLOCK_SYS_EnablePitClock** (uint32_t instance)
Enable the clock for PIT module.
- static void **CLOCK_SYS_DisablePitClock** (uint32_t instance)
Disable the clock for PIT module.
- static bool **CLOCK_SYS_GetPitGateCmd** (uint32_t instance)
Get the the clock gate state for PIT module.

- static void **CLOCK_SYS_EnableLptimerClock** (uint32_t instance)
Enable the clock for LPTIMER module.
- static void **CLOCK_SYS_DisableLptimerClock** (uint32_t instance)
Disable the clock for LPTIMER module.
- static bool **CLOCK_SYS_GetLptimerGateCmd** (uint32_t instance)
Get the the clock gate state for LPTIMER module.
- void **CLOCK_SYS_EnableSpiClock** (uint32_t instance)
Enable the clock for SPI module.
- void **CLOCK_SYS_DisableSpiClock** (uint32_t instance)
Disable the clock for SPI module.
- bool **CLOCK_SYS_GetSpiGateCmd** (uint32_t instance)
Get the the clock gate state for SPI module.
- void **CLOCK_SYS_EnableI2cClock** (uint32_t instance)
Enable the clock for I2C module.
- void **CLOCK_SYS_DisableI2cClock** (uint32_t instance)
Disable the clock for I2C module.
- bool **CLOCK_SYS_GetI2cGateCmd** (uint32_t instance)
Get the the clock gate state for I2C module.
- void **CLOCK_SYS_EnableUartClock** (uint32_t instance)
Enable the clock for UART module.
- void **CLOCK_SYS_DisableUartClock** (uint32_t instance)
Disable the clock for UART module.
- bool **CLOCK_SYS_GetUartGateCmd** (uint32_t instance)
Get the the clock gate state for UART module.
- **clock_manager_error_code_t CLOCK_SYS_Osc0Init** (osc_user_config_t *config)
Initialize OSC0.
- void **CLOCK_SYS_Osc0Deinit** (void)
De-initialize OSC0.
- static void **CLOCK_SYS_SetFtmExternalFreq** (uint32_t srcInstance, uint32_t freq)
Set the FTM external clock frequency(FTM_CLKx).
- uint32_t **CLOCK_SYS_GetRtcOutFreq** (void)
Gets RTC_CLKOUT frequency.
- static clock_rtcout_src_t **CLOCK_SYS_GetRtcOutSrc** (void)
Gets RTC_CLKOUT source.
- static void **CLOCK_SYS_SetRtcOutSrc** (clock_rtcout_src_t src)
Gets RTC_CLKOUT source.
- static uint32_t **CLOCK_SYS_GetCmtFreq** (uint32_t instance)
Gets the clock frequency for CMT module.
- static clock_usbfs_src_t **CLOCK_SYS_GetUsbfsSrc** (uint32_t instance)
Gets the clock source for USB FS OTG module.
- static void **CLOCK_SYS_SetUsbfsSrc** (uint32_t instance, clock_usbfs_src_t usbfsSrc)
Sets the clock source for USB FS OTG module.
- uint32_t **CLOCK_SYS_GetUsbfsFreq** (uint32_t instance)
Gets the clock frequency for USB FS OTG module.
- static void **CLOCK_SYS_SetUsbfsDiv** (uint32_t instance, uint8_t usbdv, uint8_t usbfrac)
Set USB FS divider setting.
- static void **CLOCK_SYS_GetUsbfsDiv** (uint32_t instance, uint8_t *usbdv, uint8_t *usbfrac)
Get USB FS divider setting.
- static clock_lpuart_src_t **CLOCK_SYS_GetLpuartSrc** (uint32_t instance)
Gets the clock source for LPUART module.
- static void **CLOCK_SYS_SetLpuartSrc** (uint32_t instance, clock_lpuart_src_t lpuartSrc)

Overview

- `uint32_t CLOCK_SYS_GetLpuartFreq (uint32_t instance)`
Gets the clock frequency for LPUART module.
- `uint32_t CLOCK_SYS_GetSaiFreq (uint32_t instance, clock_sai_src_t saiSrc)`
Gets the clock frequency for SAI.
- `static void CLOCK_SYS_EnableSaiClock (uint32_t instance)`
Enable the clock for SAI module.
- `static void CLOCK_SYS_DisableSaiClock (uint32_t instance)`
Disable the clock for SAI module.
- `static bool CLOCK_SYS_GetSaiGateCmd (uint32_t instance)`
Get the the clock gate state for SAI module.
- `static void CLOCK_SYS_EnableRtcClock (uint32_t instance)`
Enable the clock for RTC module.
- `static void CLOCK_SYS_DisableRtcClock (uint32_t instance)`
Disable the clock for RTC module.
- `static bool CLOCK_SYS_GetRtcGateCmd (uint32_t instance)`
Get the the clock gate state for RTC module.
- `static void CLOCK_SYS_EnableUsbfsClock (uint32_t instance)`
Enable the clock for USBFS module.
- `static void CLOCK_SYS_DisableUsbfsClock (uint32_t instance)`
Disable the clock for USBFS module.
- `static bool CLOCK_SYS_GetUsbfsGateCmd (uint32_t instance)`
Get the the clock gate state for USB module.
- `static void CLOCK_SYS_EnableLpuartClock (uint32_t instance)`
Enable the clock for LPUART module.
- `static void CLOCK_SYS_DisableLpuartClock (uint32_t instance)`
Disable the clock for LPUART module.
- `static bool CLOCK_SYS_GetLpuartGateCmd (uint32_t instance)`
Get the the clock gate state for LPUART module.
- `static void CLOCK_SYS_SetUsbExternalFreq (uint32_t srcInstance, uint32_t freq)`
Set the USB external clock frequency(USB_CLKIN).
- `static void CLOCK_SYS_EnableRngaClock (uint32_t instance)`
Enable the clock for RNGA module.
- `static void CLOCK_SYS_DisableRngaClock (uint32_t instance)`
Disable the clock for RNGA module.
- `static bool CLOCK_SYS_GetRngaGateCmd (uint32_t instance)`
Get the the clock gate state for RNGA module.
- `static void CLOCK_SYS_SetOutDiv3 (uint8_t outdiv3)`
Sets the clock out divider3 setting(OUTDIV3).
- `static uint8_t CLOCK_SYS_GetOutDiv3 (void)`
Gets the clock out divider3 setting(OUTDIV3).
- `uint32_t CLOCK_SYS_GetFlexbusFreq (void)`
Get flexbus clock frequency.
- `static void CLOCK_SYS_EnableFlexbusClock (uint32_t instance)`
Enable the clock for FLEXBUS module.
- `static void CLOCK_SYS_DisableFlexbusClock (uint32_t instance)`
Disable the clock for FLEXBUS module.
- `static bool CLOCK_SYS_GetFlexbusGateCmd (uint32_t instance)`
Get the the clock gate state for FLEXBUS module.
- `uint32_t CLOCK_SYS_GetFlexcanFreq (uint32_t instance, clock_flexcan_src_t flexcanSrc)`
Gets FLEXCAN clock frequency.

- `uint32_t CLOCK_SYS_GetSdhcFreq (uint32_t instance)`
Gets the clock frequency for SDHC.
- `static void CLOCK_SYS_SetSdhcSrc (uint32_t instance, clock_sdhc_src_t setting)`
Set the SDHC clock source selection.
- `static clock_sdhc_src_t CLOCK_SYS_GetSdhcSrc (uint32_t instance)`
Get the SDHC clock source selection.
- `static uint32_t CLOCK_SYS_GetUsbdcdFreq (uint32_t instance)`
Gets the clock frequency for USB DCD module.
- `static void CLOCK_SYS_EnableMpuClock (uint32_t instance)`
Enable the clock for MPU module.
- `static void CLOCK_SYS_DisableMpuClock (uint32_t instance)`
Disable the clock for MPU module.
- `static bool CLOCK_SYS_GetMpuGateCmd (uint32_t instance)`
Get the the clock gate state for MPU module.
- `static void CLOCK_SYS_EnableCmtClock (uint32_t instance)`
Enable the clock for CMT module.
- `static void CLOCK_SYS_DisableCmtClock (uint32_t instance)`
Disable the clock for CMT module.
- `static bool CLOCK_SYS_GetCmtGateCmd (uint32_t instance)`
Get the the clock gate state for CMT module.
- `static void CLOCK_SYS_EnableUsbdcdClock (uint32_t instance)`
Enable the clock for USBD_CD module.
- `static void CLOCK_SYS_DisableUsbdcdClock (uint32_t instance)`
Disable the clock for USBD_CD module.
- `static bool CLOCK_SYS_GetUsbdcdGateCmd (uint32_t instance)`
Get the the clock gate state for USBD_CD module.
- `static void CLOCK_SYS_EnableFlexcanClock (uint32_t instance)`
Enable the clock for FLEXCAN module.
- `static void CLOCK_SYS_DisableFlexcanClock (uint32_t instance)`
Disable the clock for FLEXCAN module.
- `static bool CLOCK_SYS_GetFlexcanGateCmd (uint32_t instance)`
Get the the clock gate state for FLEXCAN module.
- `static void CLOCK_SYS_EnableSdhcClock (uint32_t instance)`
Enable the clock for SDHC module.
- `static void CLOCK_SYS_DisableSdhcClock (uint32_t instance)`
Disable the clock for SDHC module.
- `static bool CLOCK_SYS_GetSdhcGateCmd (uint32_t instance)`
Get the the clock gate state for SDHC module.
- `static void CLOCK_SYS_SetSdhcExternalFreq (uint32_t srcInstance, uint32_t freq)`
Set the SDHC external clock frequency(SDHC_CLKIN).
- `static clock_rmii_src_t CLOCK_SYS_GetEnetRmiiSrc (uint32_t instance)`
Gets ethernet RMII clock source.
- `static void CLOCK_SYS_SetEnetRmiiSrc (uint32_t instance, clock_rmii_src_t rmiiSrc)`
Sets ethernet RMII clock source.
- `uint32_t CLOCK_SYS_GetEnetRmiiFreq (uint32_t instance)`
Gets ethernet RMII clock frequency.
- `static void CLOCK_SYS_SetEnetTimeStampSrc (uint32_t instance, clock_time_src_t timeSrc)`
Set the ethernet timestamp clock source selection.
- `static clock_time_src_t CLOCK_SYS_GetEnetTimeStampSrc (uint32_t instance)`
Get the ethernet timestamp clock source selection.
- `uint32_t CLOCK_SYS_GetEnetTimeStampFreq (uint32_t instance)`

Overview

- static void **CLOCK_SYS_EnableEnetClock** (uint32_t instance)
Enable the clock for ENET module.
- static void **CLOCK_SYS_DisableEnetClock** (uint32_t instance)
Disable the clock for ENET module.
- static bool **CLOCK_SYS_GetEnetGateCmd** (uint32_t instance)
Get the the clock gate state for ENET module.
- static void **CLOCK_SYS_EnableTsiClock** (uint32_t instance)
Enable the clock for TSI module.
- static void **CLOCK_SYS_DisableTsiClock** (uint32_t instance)
Disable the clock for TSI module.
- static bool **CLOCK_SYS_GetTsiGateCmd** (uint32_t instance)
Get the the clock gate state for TSI module.
- static void **CLOCK_SYS_EnableLlwuClock** (uint32_t instance)
Enable the clock for LLWU module.
- static void **CLOCK_SYS_DisableLlwuClock** (uint32_t instance)
Disable the clock for LLWU module.
- static bool **CLOCK_SYS_GetLlwuGateCmd** (uint32_t instance)
Get the the clock gate state for LLWU module.
- static void **CLOCK_SYS_SetEnetExternalFreq** (uint32_t srcInstance, uint32_t freq)
Set the ENET external clock frequency(ENET_1588_CLKIN).
- uint32_t **CLOCK_SYS_GetDmaFreq** (uint32_t instance)
Gets the clock frequency for DMA module.
- uint32_t **CLOCK_SYS_GetDmamuxFreq** (uint32_t instance)
Gets the clock frequency for DMAMUX module.
- uint32_t **CLOCK_SYS_GetPortFreq** (uint32_t instance)
Gets the clock frequency for PORT module.
- uint32_t **CLOCK_SYS_GetMpuFreq** (uint32_t instance)
Gets the clock frequency for MPU module.
- uint32_t **CLOCK_SYS_GetFlexbusFreq** (uint32_t instance)
Gets the clock frequency for FLEXBUS module.
- uint32_t **CLOCK_SYS_GetRngaFreq** (uint32_t instance)
Gets the clock frequency for RNGA module.
- uint32_t **CLOCK_SYS_GetAdcFreq** (uint32_t instance)
Gets the clock frequency for ADC module.
- uint32_t **CLOCK_SYS_GetFtmFreq** (uint32_t instance)
Gets the clock frequency for FTM module.
- uint32_t **CLOCK_SYS_GetUsbFreq** (uint32_t instance)
Gets the clock frequency for USB FS OTG module.
- uint32_t **CLOCK_SYS_GetSaiFreq** (uint32_t instance)
Gets the clock frequency for I2S module.
- static void **CLOCK_SYS_EnableUsbClock** (uint32_t instance)
Enable the clock for USBFS module.
- static void **CLOCK_SYS_DisableUsbClock** (uint32_t instance)
Disable the clock for USBFS module.
- static bool **CLOCK_SYS_GetUsbGateCmd** (uint32_t instance)
Get the the clock gate state for USB module.
- uint32_t **CLOCK_SYS_GetTpmFreq** (uint32_t instance)
Gets the clock frequency for TPM module.
- static void **CLOCK_SYS_SetTpmSrc** (uint32_t instance, clock_tpm_src_t tpmSrc)
Set the TPM clock source selection.

- static clock_tpm_src_t **CLOCK_SYS_GetTpmSrc** (uint32_t instance)
Get the TPM clock source selection.
- uint32_t **CLOCK_SYS_GetTpmExternalFreq** (uint32_t instance)
Get the TPM external clock source frequency.
- static void **CLOCK_SYS_SetTpmExternalSrc** (uint32_t instance, sim_tpm_clk_sel_t src)
Set the TPM external clock source selection.
- static sim_tpm_clk_sel_t **CLOCK_SYS_GetTpmExternalSrc** (uint32_t instance)
Set the TPM external clock source selection.
- void **CLOCK_SYS_EnableTpmClock** (uint32_t instance)
Enable the clock for TPM module.
- void **CLOCK_SYS_DisableTpmClock** (uint32_t instance)
Disable the clock for TPM module.
- bool **CLOCK_SYS_GetTpmGateCmd** (uint32_t instance)
Get the the clock gate state for TPM module.
- static void **CLOCK_SYS_SetTpmExternalFreq** (uint32_t srcInstance, uint32_t freq)
Set the TPM external clock frequency(TPM_CLKx).
- uint32_t **CLOCK_SYS_GetCopFreq** (uint32_t instance, clock_cop_src_t copSrc)
Gets the COP clock frequency.
- uint32_t **CLOCK_SYS_GetRtcFreq** (uint32_t instance, clock_er32k_src_t rtcSrc)
Gets rtc clock frequency.
- uint32_t **CLOCK_SYS_GetLpisciFreq** (uint32_t instance)
Gets the clock frequency for LPSCI module.
- static void **CLOCK_SYS_SetLpisciSrc** (uint32_t instance, clock_lpisci_src_t lpisciSrc)
Set the LPSCI clock source selection.
- static clock_lpisci_src_t **CLOCK_SYS_GetLpisciSrc** (uint32_t instance)
Get the LPSCI clock source selection.
- void **CLOCK_SYS_EnableLpisciClock** (uint32_t instance)
Enable the clock for LPSCI module.
- void **CLOCK_SYS_DisableLpisciClock** (uint32_t instance)
Disable the clock for LPSCI module.
- bool **CLOCK_SYS_GetLpisciGateCmd** (uint32_t instance)
Get the the clock gate state for LPSCI module.
- static void **CLOCK_SYS_SetOutDiv5** (uint8_t outdiv5)
Sets the clock out divider5 setting(OUTDIV5).
- static uint8_t **CLOCK_SYS_GetOutDiv5** (void)
Gets the clock out divider5 setting(OUTDIV5).
- uint32_t **CLOCK_SYS_GetOutdiv5ClockFreq** (void)
Gets the OUTDIV5 output clock for ADC.

Variables

- const uint32_t **g_simBaseAddr** []
The register base of SIM module.
- const uint32_t **g_mcgBaseAddr** []
The register base of MCG/MCG_LITE module.
- const uint32_t **g_oscBaseAddr** []
The register base of OSC module.
- **clock_manager_user_config_t g_defaultClockConfigurations** []
Default pre-defined clock configurations.

Overview

Dynamic clock setting

- `clock_manager_error_code_t CLOCK_SYS_Init (clock_manager_user_config_t const *clockConfigsPtr, uint8_t configsNumber, clock_manager_callback_user_config_t **(*callbacksPtr)[], uint8_t callbacksNumber)`
Install pre-defined clock configurations.
- `clock_manager_error_code_t CLOCK_SYS_UpdateConfiguration (uint8_t targetConfigIndex, clock_manager_policy_t policy)`
Set system clock configuration according to pre-defined structure.
- `clock_manager_error_code_t CLOCK_SYS_SetConfiguration (clock_manager_user_config_t const *config)`
Set system clock configuration.
- `uint8_t CLOCK_SYS_GetCurrentConfiguration (void)`
Get current system clock configuration.
- `clock_manager_callback_user_config_t * CLOCK_SYS_GetErrorCallback (void)`
Get the callback which returns error in last clock switch.
- `mcg_mode_error_t CLOCK_SYS_SetMcgMode (mcg_config_t const *targetConfig, void(*fullStableDelay)(void))`
Set MCG to some target mode.

42.1.1 Clock Manager

Overview

Clock Manager provides such kinds of APIs:

- Enable/Disable clock for IP modules;
- Get/Set clock source for IP modules;
- Get clock frequency for IP modules;
- Get/Set clock dividers;
- Get clock module output frequency;
- OSC initialize/deinitialize.

It covers such modules: SIM, MCG and OSC.

Clock manager also provides clock dynamic change service. Based on this service, applications could switch between pre-defined clock configurations. There is also a notification framework, drivers and applications could register callback functions to this framework, every time system clock configuration is changed, drivers and applications will receive notification and change their settings.

Enable/Disable clock for IP modules

Clock manager provides these APIs with the format:

```
/* Enable IP clock. */
void CLOCK_SYS_EnableIpClock(uint32_t instance);
/* Disable IP clock. */
void CLOCK_SYS_DisableIpClock(uint32_t instance);
/* Get IP clock status. */
bool CLOCK_SYS_GetIpGateCmd(uint32_t instance);
```

Get/Set clock source for IP modules

Clock manager only provides clock source get/set APIs for the modules whose control registers are in SIM and MCG. These APIs are defined with the format:

```
/* Get the IP module source. */
clock_ip_src_t CLOCK_SYS_GetIpSrc(uint32_t instance);

/* Set the IP module source. */
void CLOCK_SYS_SetIpSrc(uint32_t instance, clock_ip_src_t ipSrc);
```

These functions are only thin wrap of HAL drivers. The parameter `clock_ip_src_t` is enumeration defined in SIM HAL driver, please check SIM HAL driver for details.

Get clock frequency for IP modules

There are two circumstances:

1. Some IP modules' clock source are chosen by SIM registers, then clock manager provides APIs with the format:

```
uint32_t CLOCK_SYS_GetIpFreq(uint32_t instance);
```

These functions return clock frequency, if clock source is not available, will return 0.

2. Some IP modules' clock source are chosen by IP internal registers, for example SAI module. For these modules, the clock frequency APIs are defined as:

```
uint32_t CLOCK_SYS_GetIpFreq(clock_ip_src_t ipSrc, uint32_t instance);
```

The clock source selection is passed as parameter, then IP driver could read its internal register and pass it to clock manager to get clock frequency.

Get/Set clock dividers

Clock manager provides separate divider APIs for different clocks. They are defined as:

```
/* Get divider values. */
void CLOCK_SYS_GetIpDiv(uint32_t instance, type1 *div1, type2 *div2, ...);

/* Set divider values. */
void CLOCK_SYS_SetIpDiv(uint32_t instance, type1 div1, type2 div2, ...);
```

Get clock module output frequency

The clock module output clocks, including MCGFFCLK, MCGIRCLK, MCGPLLFLCLK OSCERCLK, ERCLK32K, LPO, RTC_CLKOUT, core clock, Bus clock and so on, clock manager provides separate APIs to get the frequency of these clocks. These functions are defined as:

```
uint32_t CLOCK_SYS_GetNameFreq();
```

These functions return the frequency in Hz, if clock source is not available, return 0. There is also an API [CLOCK_SYS_GetFreq\(\)](#), passing clock name as parameter could get corresponding frequency.

Overview

OSC initialize/deinitialize

Clock manager provides functions to initialize and deinitialize OSC, the OSC configurations are passed by the structure `osc_user_config_t`.

```
clock_manager_error_code_t CLOCK_SYS_Osc0Init(  
    osc_user_config_t *config);  
  
void CLOCK_SYS_Osc0Deinit(void);
```

Dynamic clock setting

Clock manager dynamic clock setting is based on two components:

1. Some pre-defined clock configurations, system could switch between these configurations dynamically.

Generally, there are two configurations, one for normal run mode and the other for very low power run (VLPR) mode. Some platforms such as K22, there is a HRUN power mode, then there will be an additional clock configuration correspondingly for peak performance. Please check the structure `g_defaultClockConfigurations` for details.

2. Notification framework.

Clock manager provides a notification mechanism, IP drivers and application should register callback functions to notification framework. To simplify, clock manager only support static callback registration. It means that in applications, all callbak functions are collected into a static table and passed to clock manager.

The clock mode transition includes 3 steps:

- (a) Before clock mode transition, clock manager will send a "BEFORE" message to callback table. When receives this message, IP drivers should check whether current work could be stopped and stop it. If the work could not be stopped, the callback function returns error. Clock manager supports two types of transition policies, graceful policy and forceful policy. When graceful policy is used, if some callbacks return error during sending "BEFORE" message, the clock mode transition stops and clock manager will send "RECOVER" message to all the drivers that have stopped. Then these drivers could recover to the previous status and continue to work. When forceful policy is used, drivers should stop forcefully.
- (b) After "BEFORE"message is processed successfully, the system clock mode will change according to the new configuration.
- (c) After system clock mode change, clock manager will send an "AFTER" message to callback table to notify drivers that clock mode transition is finished.

There is an example for how to enable dynamic clock setting in application:

```
/* Callback for UART0. */  
uart_callback_data_t uart0CallbackData;  
clock_manager_error_code_t uart0Callback(  
    clock_notify_struct_t *notify,  
    void* driverData);  
  
/* Callback for ADC1. */  
adc_callback_data_t adc1CallbackData;  
clock_manager_error_code_t adc1Callback(  
    clock_notify_struct_t *notify,  
    void* driverData);
```

```

/* Callback configuration for UART0. */
clock_manager_callback_user_config_t uart0callbackConfig =
{
    .callback      = uart0Callback,
    .callbackType = kClockManagerCallbackBeforeAfter,
    .callbackData  = &uart0CallbackData
};

/* Callback configuration for ADC1. */
clock_manager_callback_user_config_t adc1callbackConfig =
{
    .callback      = adc1Callback,
    .callbackType = kClockManagerCallbackBeforeAfter,
    .callbackData  = &adc1CallbackData
};

/* static clock callback table. */
static clock_manager_callback_user_config clockCallbackTable [] =
{
    &uart0callbackConfig,
    &adc1callbackConfig,
};

/* Set configuration and callback table to clock manager. */
CLOCK_SYS_Init(g_defaultClockConfigurations,
                CLOCK_CONFIG_NUM,
                clockCallbackTable,
                ARRAY_SIZE(clockCallbackTable));

```

Clock manager has provided the default clock configuration table with the name `g_defaultClockConfigurations`, application only need to construct the callback function table. Finally, use the function `CLOCK_SYS_Init` to setup clock configuration and callback table to clock manager.

There is an example for callback function:

```

clock_manager_error_code_t IPCallback(
    clock_notify_struct_t *notify,
                                void *callbackData)
{
    clock_manager_error_code_t ret =
        kClockManagerSuccess;

    switch (notify->notifyTpye)
    {
        case kClockManagerNotifyBefore: // Received "BEFORE" message
            if (!ready_to_change)
            {
                ret = kClockManagerError;
            }
            if ((ret == kClockManagerSuccess) ||
                (kClockManagerPolicyForcible == notify->
                policy))
            {
                // Gate IP clock and stop work.
            }
            break;
        case kClockManagerNotifyRecover: // Received "RECOVER" message
            // Recover work.
            break;
        case kClockManagerNotifyAfter: // Received "AFTER" message
            // Ungate IP clock and re-configure driver.
            break;
        default:
            ret = kClockManagerError;
    }
}

```

Data Structure Documentation

```
        break;
    }
    return ret;
}
```

42.2 Data Structure Documentation

42.2.1 struct mcg_config_t

Data Fields

- [mcg_modes_t mcg_mode](#)
MCG mode.
- [bool irclkEnable](#)
MCGIRCLK enable.
- [bool irclkEnableInStop](#)
MCGIRCLK enable in stop mode.
- [mcg_internal_ref_clock_select_t ircs](#)
MCG_C2/IRCS].
- [uint8_t fcrdiv](#)
MCG_SC[FCRDIV].
- [uint8_t frdiv](#)
MCG_C1[FRDIV].
- [mcg_dco_range_select_t drs](#)
MCG_C4[DRST_DRS].
- [mcg_dmx32_select_t dmx32](#)
MCG_C4[DMX32].

42.2.1.0.0.1 Field Documentation

42.2.1.0.0.1.1 mcg_modes_t mcg_config_t::mcg_mode

42.2.1.0.0.1.2 bool mcg_config_t::irclkEnable

42.2.1.0.0.1.3 bool mcg_config_t::irclkEnableInStop

42.2.1.0.0.1.4 mcg_internal_ref_clock_select_t mcg_config_t::ircs

42.2.1.0.0.1.5 uint8_t mcg_config_t::fcrdiv

42.2.1.0.0.1.6 uint8_t mcg_config_t::frdiv

42.2.1.0.0.1.7 mcg_dco_range_select_t mcg_config_t::drs

42.2.1.0.0.1.8 mcg_dmx32_select_t mcg_config_t::dmx32

42.2.2 struct clock_manager_user_config_t

Data Fields

- [mcg_config_t mcgConfig](#)

- `sim_config_t simConfig`
MCG configuration.
- `oscer_config_t oscerConfig`
SIM configuration.
- `oscer_config_t oscerConfig`
OSCERCLK configuration.

42.2.2.0.0.2 Field Documentation

42.2.2.0.0.2.1 `mcg_config_t clock_manager_user_config_t::mcgConfig`

42.2.2.0.0.2.2 `sim_config_t clock_manager_user_config_t::simConfig`

42.2.2.0.0.2.3 `oscer_config_t clock_manager_user_config_t::oscerConfig`

42.2.3 `struct clock_notify_struct_t`

Data Fields

- `uint8_t targetClockConfigIndex`
Target clock configuration index.
- `clock_manager_policy_t policy`
Clock transition policy.
- `clock_manager_notify_t notifyType`
Clock notification type.

42.2.3.0.0.3 Field Documentation

42.2.3.0.0.3.1 `uint8_t clock_notify_struct_t::targetClockConfigIndex`

42.2.3.0.0.3.2 `clock_manager_policy_t clock_notify_struct_t::policy`

42.2.3.0.0.3.3 `clock_manager_notify_t clock_notify_struct_t::notifyType`

42.2.4 `struct clock_manager_callback_user_config_t`

Data Fields

- `clock_manager_callback_t callback`
Entry of callback function.
- `clock_manager_callback_type_t callbackType`
Callback type.
- `void *callbackData`
Parameter of callback function.

Typedef Documentation

42.2.4.0.0.4 Field Documentation

42.2.4.0.0.4.1 `clock_manager_callback_t clock_manager_callback_user_config_t::callback`

42.2.4.0.0.4.2 `clock_manager_callback_type_t clock_manager_callback_user_config_t::callbackType`

42.2.4.0.0.4.3 `void* clock_manager_callback_user_config_t::callbackData`

42.2.5 `struct clock_manager_state_t`

Data Fields

- `clock_manager_user_config_t const * configTable`
Pointer to clock configure table.
- `uint8_t clockConfigNum`
Number of clock configurations.
- `uint8_t curConfigIndex`
Index of current configuration.
- `clock_manager_callback_user_config_t *(* callbackConfig)[]`
Pointer to callback table.
- `uint8_t callbackNum`
Number of clock callbacks.
- `uint8_t errorCallbackIndex`
Index of callback returns error.

42.2.5.0.0.5 Field Documentation

42.2.5.0.0.5.1 `clock_manager_user_config_t const* clock_manager_state_t::configTable`

42.2.5.0.0.5.2 `uint8_t clock_manager_state_t::clockConfigNum`

42.2.5.0.0.5.3 `uint8_t clock_manager_state_t::curConfigIndex`

42.2.5.0.0.5.4 `clock_manager_callback_user_config_t*(* clock_manager_state_t::callbackConfig)[]`

42.2.5.0.0.5.5 `uint8_t clock_manager_state_t::callbackNum`

42.2.5.0.0.5.6 `uint8_t clock_manager_state_t::errorCallbackIndex`

42.3 Macro Definition Documentation

42.3.1 `#define CPU_LPO_CLK_HZ 1000U`

42.4 Typedef Documentation

42.4.1 `typedef clock_manager_error_code_t(* clock_manager_callback_t)(clock_notify_struct_t *notify, void *callbackData)`

42.5 Enumeration Type Documentation

42.5.1 enum clock_names_t

Enumerator

- kCoreClock* Core clock.
- kSystemClock* System clock.
- kPlatformClock* Platform clock.
- kBusClock* Bus clock.
- kFlexBusClock* FlexBus clock.
- kFlashClock* Flash clock.
- kOsc32kClock* ERCLK32K.
- kOsc0ErClock* OSC0ERCLK.
- kOsc1ErClock* OSC1ERCLK.
- kOsc0ErClockUndiv* OSC0ERCLK_UNDIV.
- kIrc48mClock* IRC 48M.
- kRtcoutClock* RTC_CLKOUT.
- kMcgFfClock* MCG fixed frequency clock (MCGFFCLK)
- kMcgFllClock* MCGFLLCLK.
- kMcgPll0Clock* MCGPLL0CLK.
- kMcgPll1Clock* MCGPLL1CLK.
- kMcgOutClock* MCGOUTCLK.
- kMcgIrClock* MCGIRCLK.
- kLpoClock* LPO clock.

42.5.2 enum clock_manager_error_code_t

Enumerator

- kClockManagerSuccess* Success.
- kClockManagerError* Some error occurs.
- kClockManagerNoSuchClockName* Invalid name.
- kClockManagerInvalidParam* Invalid parameter.
- kClockManagerErrorOutOfRange* Configuration index out of range.
- kClockManagerErrorNotificationBefore* Error occurs during send "BEFORE" notification.
- kClockManagerErrorNotificationAfter* Error occurs during send "AFTER" notification.
- kClockManagerErrorUnknown* Unknown error.

42.5.3 enum clock_manager_notify_t

Enumerator

- kClockManagerNotifyRecover* Notify IP to recover to previous work state.

Function Documentation

kClockManagerNotifyBefore Notify IP that system will change clock setting.

kClockManagerNotifyAfter Notify IP that have changed to new clock setting.

42.5.4 enum clock_manager_callback_type_t

Enumerator

kClockManagerCallbackBefore Callback handles BEFORE notification.

kClockManagerCallbackAfter Callback handles AFTER notification.

kClockManagerCallbackBeforeAfter Callback handles BEFORE and AFTER notification.

42.5.5 enum clock_manager_policy_t

Enumerator

kClockManagerPolicyAgreement Clock transfers gracefully.

kClockManagerPolicyForcible Clock transfers forcefully.

42.6 Function Documentation

42.6.1 clock_manager_error_code_t CLOCK_SYS_GetFreq (clock_names_t *clockName*, uint32_t * *frequency*)

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in *clock_names_t*. The MCG must be properly configured before using this function. See the reference manual for supported clock names for different chip families. The returned value is in Hertz. If it cannot find the clock name or the name is not supported for a specific chip family, it returns an error.

Parameters

<i>clockName</i>	Clock names defined in <i>clock_names_t</i>
<i>frequency</i>	Returned clock frequency value in Hertz

Returns

status Error code defined in *clock_manager_error_code_t*

42.6.2 uint32_t CLOCK_SYS_GetCoreClockFreq (void)

This function gets the core clock frequency.

Returns

Current core clock frequency.

42.6.3 uint32_t CLOCK_SYS_GetSystemClockFreq (void)

This function gets the system clock frequency.

Returns

Current system clock frequency.

42.6.4 uint32_t CLOCK_SYS_GetBusClockFreq (void)

This function gets the bus clock frequency.

Returns

Current bus clock frequency.

42.6.5 uint32_t CLOCK_SYS_GetFlashClockFreq (void)

This function gets the flash clock frequency.

Returns

Current flash clock frequency.

42.6.6 static uint32_t CLOCK_SYS_GetLpoClockFreq (void) [inline], [static]

This function gets the LPO clock frequency.

Returns

Current clock frequency.

Function Documentation

42.6.7 `clock_manager_error_code_t CLOCK_SYS_Init (clock_manager_user_config_t const * clockConfigsPtr, uint8_t configsNumber, clock_manager_callback_user_config_t **() callbacksPtr[], uint8_t callbacksNumber)`

This function installs the pre-defined clock configuration table to clock manager.

Parameters

<i>clockConfigs- Ptr</i>	Pointer to the clock configuration table.
<i>configsNumber</i>	Number of clock configurations in table.
<i>callbacksPtr</i>	Pointer to the callback configuration table.
<i>callbacks- Number</i>	Number of callback configurations in table.

Returns

Error code.

42.6.8 **clock_manager_error_code_t CLOCK_SYS_UpdateConfiguration (uint8_t targetConfigIndex, clock_manager_policy_t policy)**

This function sets system to target clock configuration, before transition, clock manager will send notifications to all drivers registered to the callback table. When graceful policy is used, if some drivers are not ready to change, clock transition will not occur, all drivers still work in previous configuration and error is returned. When forceful policy is used, all drivers should stop work and system changes to new clock configuration.

Parameters

<i>targetConfig- Index</i>	Index of the clock configuration.
<i>policy</i>	Transaction policy, graceful or forceful.

Returns

Error code.

Note

If external clock is used in the target mode, please make sure it is enabled, for example, if the external oscillator is used, please setup EREFS/HGO correctly and make sure OSCINIT is set.

42.6.9 **clock_manager_error_code_t CLOCK_SYS_SetConfiguration (clock_manager_user_config_t const * config)**

This function sets the system to target configuration, it only sets the clock modules registers for clock mode change, but not send notifications to drivers. This function is different by different SoCs.

Function Documentation

Parameters

<i>config</i>	Target configuration.
---------------	-----------------------

Returns

Error code.

Note

If external clock is used in the target mode, please make sure it is enabled, for example, if the external oscillator is used, please setup EREFS/HGO correctly and make sure OSCINIT is set.

42.6.10 uint8_t CLOCK_SYS_GetCurrentConfiguration (void)

Returns

Current clock configuration index.

42.6.11 clock_manager_callback_user_config_t* CLOCK_SYS_GetErrorCallback (void)

When graceful policy is used, if some IP is not ready to change clock setting, the callback will return error and system stay in current configuration. Applications can use this function to check which IP callback returns error.

Returns

Pointer to the callback which returns error.

42.6.12 mcg_mode_error_t CLOCK_SYS_SetMcgMode (mcg_config_t const * targetConfig, void(*)(void) fllStableDelay)

This function sets MCG to some target mode defined by the configure structure, if can not switch to target mode directly, this function will choose the proper path.

Parameters

<i>targetConfig</i>	Pointer to the target MCG mode configuration structure.
<i>fllStableDelay</i>	Delay function to make sure FLL is stable.

Returns

Error code.

Note

If external clock is used in the target mode, please make sure it is enabled, for example, if the external oscillator is used, please setup EREFS/HGO correctly and make sure OSCINIT is set.

42.6.13 static void CLOCK_SYS_SetOutDiv1 (uint8_t *outdiv1*) [inline], [static]

This function sets divide value OUTDIV1.

Parameters

<i>outdiv1</i>	Outdivider1 setting
----------------	---------------------

42.6.14 static uint8_t CLOCK_SYS_GetOutDiv1 (void) [inline], [static]

This function gets divide value OUTDIV1.

Returns

Outdivider1 setting

42.6.15 static void CLOCK_SYS_SetOutDiv2 (uint8_t *outdiv2*) [inline], [static]

This function sets divide value OUTDIV2.

Function Documentation

Parameters

<i>outdiv2</i>	Outdivider2 setting
----------------	---------------------

42.6.16 static uint8_t CLOCK_SYS_GetOutDiv2(void) [inline], [static]

This function gets divide value OUTDIV2.

Returns

Outdivider2 setting

42.6.17 static void CLOCK_SYS_SetOutDiv4(uint8_t *outdiv4*) [inline], [static]

This function sets divide value OUTDIV4.

Parameters

<i>outdiv4</i>	Outdivider4 setting
----------------	---------------------

42.6.18 static uint8_t CLOCK_SYS_GetOutDiv4(void) [inline], [static]

This function gets divide value OUTDIV4.

Returns

Outdivider4 setting

42.6.19 static void CLOCK_SYS_SetOutDiv(uint8_t *outdiv1*, uint8_t *outdiv2*, uint8_t *outdiv3*, uint8_t *outdiv4*) [inline], [static]

This function sets the setting for all clock out dividers at the same time.

Parameters

<i>outdiv1</i>	Outdivider1 setting
<i>outdiv2</i>	Outdivider2 setting
<i>outdiv3</i>	Outdivider3 setting
<i>outdiv4</i>	Outdivider4 setting

42.6.20 static void CLOCK_SYS_GetOutDiv (*uint8_t * outdiv1, uint8_t * outdiv2, uint8_t * outdiv3, uint8_t * outdiv4*) [inline], [static]

This function gets the setting for all clock out dividers at the same time.

Parameters

<i>outdiv1</i>	Outdivider1 setting
<i>outdiv2</i>	Outdivider2 setting
<i>outdiv3</i>	Outdivider3 setting
<i>outdiv4</i>	Outdivider4 setting

42.6.21 uint32_t CLOCK_SYS_GetPllfllClockFreq (void)

This function gets the frequency of the MCGPLLCLK/MCGFLLCLK/IRC48MCLK.

Returns

Current clock frequency.

42.6.22 static void CLOCK_SYS_SetPllfllSel (*clock_pllifll_sel_t setting*) [inline], [static]

This function sets the selection of the high frequency clock for various peripheral clocking options

Parameters

Function Documentation

<i>setting</i>	The value to set.
----------------	-------------------

42.6.23 **static clock_pllfill_sel_t CLOCK_SYS_GetPllfillSel (void) [inline], [static]**

This function gets the selection of the high frequency clock for various peripheral clocking options

Returns

Current selection.

42.6.24 **static uint32_t CLOCK_SYS_GetFixedFreqClockFreq (void) [inline], [static]**

This function gets the MCG fixed frequency clock (MCGFFCLK) frequency.

Returns

Current clock frequency.

42.6.25 **static uint32_t CLOCK_SYS_GetInternalRefClockFreq (void) [inline], [static]**

This function gets the internal reference clock frequency.

Returns

Current clock frequency.

42.6.26 **uint32_t CLOCK_SYS_GetExternalRefClock32kFreq (void)**

This function gets the external reference (ERCLK32K) clock frequency.

Returns

Current frequency.

**42.6.27 static void CLOCK_SYS_SetExternalRefClock32kSrc (`clock_er32k_src_t`
`src`) [inline], [static]**

This function sets the clock selection of ERCLK32K.

Function Documentation

Parameters

<code>src</code>	clock source.
------------------	---------------

42.6.28 static clock_er32k_src_t CLOCK_SYS_GetExternalRefClock32kSrc (void) [inline], [static]

This function gets the clock selection of ERCLK32K.

Returns

Current selection.

42.6.29 uint32_t CLOCK_SYS_GetOsc0ExternalRefClockFreq (void)

This function gets the OSC0 external reference frequency.

Returns

Current frequency.

42.6.30 uint32_t CLOCK_SYS_GetOsc0ExternalRefClockUndivFreq (void)

This function gets the undivided OSC0 external reference frequency.

Returns

Current frequency.

42.6.31 uint32_t CLOCK_SYS_GetWdogFreq (uint32_t *instance*, clock_wdog_src_t *wdogSrc*)

This function gets the watch dog clock frequency.

Parameters

<i>instance</i>	module device instance
<i>wdogSrc</i>	watch dog clock source selection, WDOG_STCTRLH[CLKSRC].

Returns

Current frequency.

42.6.32 static clock_trace_src_t CLOCK_SYS_GetTraceSrc (uint32_t *instance*) [inline], [static]

This function gets the debug trace clock source.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Current source.

42.6.33 static void CLOCK_SYS_SetTraceSrc (uint32_t *instance*, clock_trace_src_t src) [inline], [static]

This function sets the debug trace clock source.

Parameters

<i>instance</i>	module device instance.
<i>src</i>	debug trace clock source.

42.6.34 uint32_t CLOCK_SYS_GetTraceFreq (uint32_t *instance*)

This function gets the debug trace clock frequency.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Current frequency.

This function gets the debug trace clock frequency.

Parameters

<i>instance</i>	module device instance.
-----------------	-------------------------

Returns

Current frequency.

42.6.35 `uint32_t CLOCK_SYS_GetPortFilterFreq (uint32_t instance, clock_port_filter_src_t src)`

This function gets the PORTx digital input filter clock frequency.

Parameters

<i>instance</i>	module device instance.
<i>src</i>	PORTx source selection.

Returns

Current frequency.

42.6.36 `uint32_t CLOCK_SYS_GetLptmrFreq (uint32_t instance, clock_lptmr_src_t lptmrSrc)`

This function gets the LPTMRx pre-scaler/glitch filter clock frequency.

Parameters

<i>instance</i>	module device instance
<i>lptmrSrc</i>	LPTMRx clock source selection.

Returns

Current frequency.

42.6.37 static uint32_t CLOCK_SYS_GetEwmFreq (uint32_t *instance*) [inline], [static]

This function gets the clock frequency for EWM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for EWM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.38 static uint32_t CLOCK_SYS_GetFtfFreq (uint32_t *instance*) [inline], [static]

(Flash Memory)

This function gets the clock frequency for FTF module. (Flash Memory)

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

(Flash Memory)

This function gets the clock frequency for FTF module. (Flash Memory)

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.39 static uint32_t CLOCK_SYS_GetCrcFreq(uint32_t *instance*) [inline], [static]

This function gets the clock frequency for CRC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for CRC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

**42.6.40 static uint32_t CLOCK_SYS_GetCmpFreq (uint32_t *instance*)
[inline], [static]**

This function gets the clock frequency for CMP module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for CMP module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.41 static uint32_t CLOCK_SYS_GetVrefFreq(uint32_t *instance*) [inline], [static]

This function gets the clock frequency for VREF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for VREF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.42 static uint32_t CLOCK_SYS_GetPdbFreq(uint32_t *instance*) [inline], [static]

This function gets the clock frequency for PDB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for PDB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.43 static uint32_t CLOCK_SYS_GetPitFreq (uint32_t *instance*) [inline], [static]

This function gets the clock frequency for PIT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for PIT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.44 static uint32_t CLOCK_SYS_GetSpiFreq (uint32_t *instance*) [inline], [static]

This function gets the clock frequency for SPI module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for SPI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.45 static uint32_t CLOCK_SYS_GetI2cFreq (uint32_t *instance*) [inline], [static]

This function gets the clock frequency for I2C module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for I2C module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.46 static uint32_t CLOCK_SYS_GetAdcAltFreq (uint32_t *instance*) [inline], [static]

This function gets the ADC alternate clock frequency.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq Current frequency.

42.6.47 static uint32_t CLOCK_SYS_GetAdcAlt2Freq (uint32_t *instance*) [inline], [static]

This function gets the ADC alternate 2 clock frequency (ALTCLK2).

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq Current frequency.

42.6.48 static uint32_t CLOCK_SYS_GetFtmFixedFreq (uint32_t *instance*) [inline], [static]

This function gets the FTM fixed frequency clock frequency.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq Current frequency.

42.6.49 uint32_t CLOCK_SYS_GetFtmExternalFreq (uint32_t *instance*)

This function gets the FTM external clock frequency.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq Current frequency.

42.6.50 static sim_ftm_clk_sel_t CLOCK_SYS_GetFtmExternalSrc (uint32_t *instance*) [inline], [static]

This function gets the FTM external clock source.

Parameters

<i>instance</i>	module device instance.
-----------------	-------------------------

Returns

Ftm external clock source.

42.6.51 static void CLOCK_SYS_SetFtmExternalSrc (uint32_t *instance*, sim_ftm_clk_sel_t *ftmSrc*) [inline], [static]

This function sets the FTM external clock source.

Parameters

<i>instance</i>	module device instance.
<i>ftmSrc</i>	FTM clock source setting.

42.6.52 uint32_t CLOCK_SYS_GetUartFreq (uint32_t *instance*)

Get the UART clock frequency.

This function gets the clock frequency for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the UART clock frequency.

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

Current frequency.

42.6.53 static uint32_t CLOCK_SYS_GetGpioFreq (uint32_t *instance*) [inline], [static]

This function gets the clock frequency for GPIO module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for GPIO module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.54 static void CLOCK_SYS_EnableDmaClock (uint32_t *instance*) [inline], [static]

This function enables the clock for DMA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.55 static void CLOCK_SYS_DisableDmaClock (uint32_t *instance*) [inline], [static]

This function disables the clock for DMA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.56 static bool CLOCK_SYS_GetDmaGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for DMA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.57 static void CLOCK_SYS_EnableDmamuxClock (uint32_t *instance*)
[inline], [static]**

This function enables the clock for DMAMUX module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.58 static void CLOCK_SYS_DisableDmamuxClock (uint32_t *instance*) [inline], [static]

This function disables the clock for DMAMUX module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.59 static bool CLOCK_SYS_GetDmamuxGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for DMAMUX module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.60 void CLOCK_SYS_EnablePortClock (uint32_t *instance*) [inline]

This function enables the clock for PORT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.61 void CLOCK_SYS_DisablePortClock (uint32_t *instance*) [inline]

This function disables the clock for PORT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.62 **bool CLOCK_SYS_GetPortGateCmd (uint32_t *instance*) [inline]**

This function will get the clock gate state for PORT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for PORT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.63 **static void CLOCK_SYS_EnableEwmClock (uint32_t *instance*) [inline], [static]**

This function enables the clock for EWM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.64 **static void CLOCK_SYS_DisableEwmClock (uint32_t *instance*) [inline], [static]**

This function disables the clock for EWM module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.65 static bool CLOCK_SYS_GetEwmGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for EWM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.66 static void CLOCK_SYS_EnableFtfClock (uint32_t *instance*) [inline], [static]

This function enables the clock for FTF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.67 static void CLOCK_SYS_DisableFtfClock (uint32_t *instance*) [inline], [static]

This function disables the clock for FTF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.68 static bool CLOCK_SYS_GetFtfGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for FTF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.69 static void CLOCK_SYS_EnableCrcClock (uint32_t *instance*) [inline], [static]

This function enables the clock for CRC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.70 static void CLOCK_SYS_DisableCrcClock (uint32_t *instance*) [inline], [static]

This function disables the clock for CRC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.71 static bool CLOCK_SYS_GetCrcGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for CRC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

Function Documentation

42.6.72 void CLOCK_SYS_EnableAdcClock(uint32_t *instance*) [inline]

This function enables the clock for ADC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.73 void CLOCK_SYS_DisableAdcClock(uint32_t *instance*) [inline]

This function disables the clock for ADC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.74 bool CLOCK_SYS_GetAdcGateCmd(uint32_t *instance*) [inline]

This function will get the clock gate state for ADC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for ADC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.75 static void CLOCK_SYS_EnableCmpClock(uint32_t *instance*) [inline], [static]

This function enables the clock for CMP module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.76 static void CLOCK_SYS_DisableCmpClock (uint32_t *instance*) [inline], [static]

This function disables the clock for CMP module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.77 static bool CLOCK_SYS_GetCmpGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for CMP module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.78 void CLOCK_SYS_EnableDacClock (uint32_t *instance*) [inline]

This function enables the clock for DAC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.79 void CLOCK_SYS_DisableDacClock (uint32_t *instance*) [inline]

This function disables the clock for DAC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.80 **bool CLOCK_SYS_GetDacGateCmd (uint32_t *instance*) [inline]**

This function will get the clock gate state for DAC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for DAC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.81 **static void CLOCK_SYS_EnableVrefClock (uint32_t *instance*) [inline], [static]**

This function enables the clock for VREF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.82 **static void CLOCK_SYS_DisableVrefClock (uint32_t *instance*) [inline], [static]**

This function disables the clock for VREF module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.83 static bool CLOCK_SYS_GetVrefGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for VREF module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.84 static void CLOCK_SYS_EnablePdbClock (uint32_t *instance*) [inline], [static]

This function enables the clock for PDB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.85 static void CLOCK_SYS_DisablePdbClock (uint32_t *instance*) [inline], [static]

This function disables the clock for PDB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.86 static bool CLOCK_SYS_GetPdbGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for PDB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.87 void CLOCK_SYS_EnableFtmClock (uint32_t *instance*) [inline]

This function enables the clock for FTM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.88 void CLOCK_SYS_DisableFtmClock (uint32_t *instance*) [inline]

This function disables the clock for FTM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.89 bool CLOCK_SYS_GetFtmGateCmd (uint32_t *instance*) [inline]

This function will get the clock gate state for FTM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.90 static void CLOCK_SYS_EnablePitClock (uint32_t *instance*) [inline], [static]

This function enables the clock for PIT module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.91 static void CLOCK_SYS_DisablePitClock (uint32_t *instance*) [inline], [static]

This function disables the clock for PIT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.92 static bool CLOCK_SYS_GetPitGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for PIT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.93 static void CLOCK_SYS_EnableLptimerClock (uint32_t *instance*) [inline], [static]

This function enables the clock for LPTIMER module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.94 static void CLOCK_SYS_DisableLptimerClock (uint32_t *instance*) [inline], [static]

This function disables the clock for LPTIMER module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.95 static bool CLOCK_SYS_GetLptimerGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for LPTIMER module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.96 void CLOCK_SYS_EnableSpiClock (uint32_t *instance*) [inline]

This function enables the clock for SPI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.97 void CLOCK_SYS_DisableSpiClock (uint32_t *instance*) [inline]

This function disables the clock for SPI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.98 bool CLOCK_SYS_GetSpiGateCmd (uint32_t *instance*) [inline]

This function will get the clock gate state for SPI module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for SPI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.99 void CLOCK_SYS_EnableI2cClock (uint32_t *instance*) [inline]

This function enables the clock for I2C module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.100 void CLOCK_SYS_DisableI2cClock (uint32_t *instance*) [inline]

This function disables the clock for I2C module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.101 bool CLOCK_SYS_GetI2cGateCmd (uint32_t *instance*) [inline]

This function will get the clock gate state for I2C module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for I2C module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.102 void CLOCK_SYS_EnableUartClock (uint32_t *instance*) [inline]

This function enables the clock for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.103 void CLOCK_SYS_DisableUartClock (uint32_t *instance*) [inline]

This function disables the clock for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.104 bool CLOCK_SYS_GetUartGateCmd (uint32_t *instance*) [inline]

This function will get the clock gate state for UART module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function gets the clock gate state for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.105 **clock_manager_error_code_t CLOCK_SYS_Osc0Init (osc_user_config_t * config)**

This function initializes OSC0 according to board configuration.

Parameters

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

Returns

kClockManagerSuccess on success.

42.6.106 **void CLOCK_SYS_Osc0Deinit (void)**

Deinitialize OSC0.

This function de-initializes OSC0.

This function deinitializes OSC0.

42.6.107 **static void CLOCK_SYS_SetFtmExternalFreq (uint32_t srcInstance, uint32_t freq) [inline], [static]**

This function sets the FTM external clock frequency (FTM_CLKx).

Parameters

<i>srcInstance</i>	External source instance.
<i>freq</i>	Frequency value.

42.6.108 **uint32_t CLOCK_SYS_GetRtcOutFreq (void)**

This function gets the frequency of RTC_CLKOUT.

Returns

Current frequency.

42.6.109 **static clock_rtcout_src_t CLOCK_SYS_GetRtcOutSrc (void) [inline], [static]**

This function gets the source of RTC_CLKOUT. It is determined by RTCCLKOUTSEL.

Returns

Current source.

42.6.110 **static void CLOCK_SYS_SetRtcOutSrc (clock_rtcout_src_t src) [inline], [static]**

This function sets the source of RTC_CLKOUT.

Parameters

<i>src</i>	RTC_CLKOUT source to set.
------------	---------------------------

42.6.111 **static uint32_t CLOCK_SYS_GetCmtFreq (uint32_t instance) [inline], [static]**

This function gets the clock frequency for CMT module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for CMT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

**42.6.112 static `clock_usbfs_src_t` `CLOCK_SYS_GetUsbfsSrc` (`uint32_t instance`)
[inline], [static]**

This function gets the clock source for USB FS OTG module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Clock source.

**42.6.113 static void `CLOCK_SYS_SetUsbfsSrc` (`uint32_t instance,`
`clock_usbfs_src_t usbfsSrc`) [inline], [static]**

This function sets the clock source for USB FS OTG module.

Parameters

<i>instance</i>	module device instance.
<i>usbfsSrc</i>	USB FS clock source setting.

42.6.114 uint32_t CLOCK_SYS_GetUsbfsFreq (uint32_t *instance*)

This function gets the clock frequency for USB FS OTG module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.115 static void CLOCK_SYS_SetUsbfsDiv (uint32_t *instance*, uint8_t *usbdiv*, uint8_t *usbfrac*) [inline], [static]

This function sets USB FS divider setting. Divider output clock = Divider input clock * [(USBFSFRA-C+1) / (USBFSDIV+1)]

Parameters

<i>instance</i>	USB FS module instance.
<i>usbdiv</i>	Value of USBFSDIV.
<i>usbfrac</i>	Value of USBFSFRAC.

42.6.116 static void CLOCK_SYS_GetUsbfsDiv (uint32_t *instance*, uint8_t * *usbdiv*, uint8_t * *usbfrac*) [inline], [static]

This function gets USB FS divider setting. Divider output clock = Divider input clock * [(USBFSFRA-C+1) / (USBFSDIV+1)]

Parameters

Function Documentation

<i>instance</i>	USB FS module instance.
<i>usbdv</i>	Value of USBFSDIV.
<i>usbfrac</i>	Value of USBFSFRAC.

42.6.117 static clock_lpuart_src_t CLOCK_SYS_GetLpuartSrc (uint32_t *instance*) [inline], [static]

This function gets the clock source for LPUART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Clock source.

42.6.118 static void CLOCK_SYS_SetLpuartSrc (uint32_t *instance*, clock_lpuart_src_t *lpuartSrc*) [inline], [static]

This function sets the clock source for LPUART module.

Parameters

<i>instance</i>	module device instance
<i>lpuartSrc</i>	Clock source.

42.6.119 uint32_t CLOCK_SYS_GetLpuartFreq (uint32_t *instance*)

This function gets the clock frequency for LPUART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.120 `uint32_t CLOCK_SYS_GetSaiFreq (uint32_t instance, clock_sai_src_t saiSrc)`

This function gets the clock frequency for SAI.

Function Documentation

Parameters

<i>instance</i>	module device instance.
<i>saiSrc</i>	SAI clock source selection according to its internal register.

Returns

freq clock frequency for this module.

42.6.121 static void CLOCK_SYS_EnableSaiClock (uint32_t *instance*) [inline], [static]

This function enables the clock for SAI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.122 static void CLOCK_SYS_DisableSaiClock (uint32_t *instance*) [inline], [static]

This function disables the clock for SAI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.123 static bool CLOCK_SYS_GetSaiGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for SAI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.124 static void CLOCK_SYS_EnableRtcClock (uint32_t *instance*)
[inline], [static]**

This function enables the clock for RTC module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.125 static void CLOCK_SYS_DisableRtcClock (uint32_t *instance*)
[inline], [static]**

This function disables the clock for RTC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.126 static bool CLOCK_SYS_GetRtcGateCmd (uint32_t *instance*)
[inline], [static]**

This function will get the clock gate state for RTC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.127 static void CLOCK_SYS_EnableUsbfsClock (uint32_t *instance*)
[inline], [static]**

This function enables the clock for USBFS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.128 static void CLOCK_SYS_DisableUsbfsClock (uint32_t *instance*)
[inline], [static]**

This function disables the clock for USBFS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.129 static bool CLOCK_SYS_GetUsbfsGateCmd (uint32_t *instance*)
[inline], [static]**

This function will get the clock gate state for USB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.130 static void CLOCK_SYS_EnableLpuartClock (uint32_t *instance*)
[inline], [static]**

Enable the clock for UART module.

This function enables the clock for LPUART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

This function enables the clock for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.131 static void CLOCK_SYS_DisableLpuartClock (uint32_t *instance*)
[inline], [static]**

Disable the clock for UART module.

This function disables the clock for LPUART module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

This function disables the clock for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.132 static bool CLOCK_SYS_GetLpuartGateCmd (uint32_t *instance*) [inline], [static]

Get the the clock gate state for UART module.

This function will get the clock gate state for LPUART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function will get the clock gate state for UART module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.133 static void CLOCK_SYS_SetUsbExternalFreq (uint32_t *srcInstance*, uint32_t *freq*) [inline], [static]

This function sets the USB external clock frequency (USB_CLKIN).

Parameters

<i>srcInstance</i>	External source instance.
<i>freq</i>	Frequency value.

42.6.134 static void CLOCK_SYS_EnableRngaClock (uint32_t *instance*) [inline], [static]

This function enables the clock for RNGA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.135 static void CLOCK_SYS_DisableRngaClock (uint32_t *instance*) [inline], [static]

This function disables the clock for RNGA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.136 static bool CLOCK_SYS_GetRngaGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for RNGA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.137 static void CLOCK_SYS_SetOutDiv3 (uint8_t *outdiv3*) [inline], [static]

This function sets divide value OUTDIV3.

Function Documentation

Parameters

<i>outdiv3</i>	Outdivider3 setting
----------------	---------------------

42.6.138 static uint8_t CLOCK_SYS_GetOutDiv3(void) [inline], [static]

This function gets divide value OUTDIV3.

Returns

Outdivider3 setting

42.6.139 uint32_t CLOCK_SYS_GetFlexbusFreq(void)

This function gets the flexbus clock frequency.

Returns

Current flexbus clock frequency.

42.6.140 static void CLOCK_SYS_EnableFlexbusClock(uint32_t *instance*) [inline], [static]

This function enables the clock for FLEXBUS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.141 static void CLOCK_SYS_DisableFlexbusClock(uint32_t *instance*) [inline], [static]

This function disables the clock for FLEXBUS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.142 static bool CLOCK_SYS_GetFlexbusGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for FLEXBUS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.143 uint32_t CLOCK_SYS_GetFlexcanFreq (uint32_t *instance*, clock_flexcan_src_t *flexcanSrc*)

This function gets the FLEXCAN clock frequency.

Parameters

<i>instance</i>	module device instance
<i>flexcanSrc</i>	clock source

Returns

Current frequency.

42.6.144 uint32_t CLOCK_SYS_GetSdhcFreq (uint32_t *instance*)

Gets the clock frequency for SDHC module.

This function gets the clock frequency for SDHC.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module.

This function gets the clock frequency for SDHC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for SDHC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

**42.6.145 static void CLOCK_SYS_SetSdhcSrc (uint32_t *instance*,
clock_sdhc_src_t *setting*) [inline], [static]**

This function sets the SDHC clock source selection.

Parameters

<i>instance</i>	IP instance.
<i>setting</i>	The value to set.

**42.6.146 static clock_sdhc_src_t CLOCK_SYS_GetSdhcSrc (uint32_t *instance*)
[inline], [static]**

This function gets the SDHC clock source selection.

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

Current selection.

42.6.147 static uint32_t CLOCK_SYS_GetUsbdcdFreq (uint32_t *instance*) [inline], [static]

This function gets the clock frequency for USB DCD module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for USB DCD module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.148 static void CLOCK_SYS_EnableMpuClock (uint32_t *instance*) [inline], [static]

This function enables the clock for MPU module.

Parameters

Function Documentation

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.149 static void CLOCK_SYS_DisableMpuClock (uint32_t *instance*)
[inline], [static]**

This function disables the clock for MPU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.150 static bool CLOCK_SYS_GetMpuGateCmd (uint32_t *instance*)
[inline], [static]**

This function will get the clock gate state for MPU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.151 static void CLOCK_SYS_EnableCmtClock (uint32_t *instance*)
[inline], [static]**

This function enables the clock for CMT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.152 static void CLOCK_SYS_DisableCmtClock (uint32_t *instance*)
[inline], [static]**

This function disables the clock for CMT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.153 static bool CLOCK_SYS_GetCmtGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for CMT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.154 static void CLOCK_SYS_EnableUsbdcClock (uint32_t *instance*) [inline], [static]

This function enables the clock for USBDCD module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.155 static void CLOCK_SYS_DisableUsbdcClock (uint32_t *instance*) [inline], [static]

This function disables the clock for USBDCD module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.156 static bool CLOCK_SYS_GetUsbdcGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for USBDCD module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.157 static void CLOCK_SYS_EnableFlexcanClock (uint32_t *instance*) [inline], [static]

This function enables the clock for FLEXCAN module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.158 static void CLOCK_SYS_DisableFlexcanClock (uint32_t *instance*) [inline], [static]

This function disables the clock for FLEXCAN module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.159 static bool CLOCK_SYS_GetFlexcanGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for FLEXCAN module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.160 static void CLOCK_SYS_EnableSdhcClock (uint32_t *instance*)
[inline], [static]**

This function enables the clock for SDHC module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.161 static void CLOCK_SYS_DisableSdhcClock (uint32_t *instance*)
[inline], [static]**

This function disables the clock for SDHC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.162 static bool CLOCK_SYS_GetSdhcGateCmd (uint32_t *instance*)
[inline], [static]**

This function will get the clock gate state for SDHC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.163 static void CLOCK_SYS_SetSdhcExternalFreq (uint32_t *srcInstance*,
uint32_t *freq*) [inline], [static]**

This function sets the SDHC external clock frequency(SDHC_CLKIN).

Parameters

<i>srcInstance</i>	External source instance.
<i>freq</i>	Frequency value.

42.6.164 static clock_rmii_src_t CLOCK_SYS_GetEnetRmiiSrc (uint32_t *instance*) [inline], [static]

This function gets the ethernet RMII clock source.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Current source.

42.6.165 static void CLOCK_SYS_SetEnetRmiiSrc (uint32_t *instance*, clock_rmii_src_t *rmiiSrc*) [inline], [static]

This function sets the ethernet RMII clock source.

Parameters

<i>instance</i>	module device instance.
<i>rmiiSrc</i>	RMII clock source.

42.6.166 uint32_t CLOCK_SYS_GetEnetRmiiFreq (uint32_t *instance*)

Gets the clock frequency for ENET module RMII clock.

This function gets the ethernet RMII clock frequency.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Current frequency.

This function gets the clock frequency for ENET module RMII clock..

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for ENET module RMII clock.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.167 static void CLOCK_SYS_SetEnetTimeStampSrc (uint32_t *instance*, clock_time_src_t *timeSrc*) [inline], [static]

This function sets the ethernet timestamp clock source selection.

Parameters

<i>instance</i>	module device instance.
<i>timeSrc</i>	Ethernet timestamp clock source.

42.6.168 static clock_time_src_t CLOCK_SYS_GetEnetTimeStampSrc (uint32_t *instance*) [inline], [static]

This function gets the ethernet timestamp clock source selection.

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

Current source.

42.6.169 uint32_t CLOCK_SYS_GetEnetTimeStampFreq (uint32_t *instance*)

Gets the clock frequency for ENET module TIME clock.

This function gets the ethernet timestamp clock frequency.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Current frequency.

This function gets the clock frequency for ENET module TIME clock..

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for ENET module TIME clock.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.170 static void CLOCK_SYS_EnableEnetClock (uint32_t *instance*) [inline], [static]

This function enables the clock for ENET module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.171 static void CLOCK_SYS_DisableEnetClock (uint32_t *instance*) [inline], [static]

This function disables the clock for ENET module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.172 static bool CLOCK_SYS_GetEnetGateCmd (uint32_t *instance*)
[inline], [static]**

This function will get the clock gate state for ENET module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

**42.6.173 static void CLOCK_SYS_EnableTsiClock (uint32_t *instance*)
[inline], [static]**

This function enables the clock for TSI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.174 static void CLOCK_SYS_DisableTsiClock (uint32_t *instance*)
[inline], [static]**

This function disables the clock for TSI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

**42.6.175 static bool CLOCK_SYS_GetTsiGateCmd (uint32_t *instance*)
[inline], [static]**

This function will get the clock gate state for TSI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.176 static void CLOCK_SYS_EnableLlwuClock (uint32_t *instance*) [inline], [static]

This function enables the clock for LLWU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.177 static void CLOCK_SYS_DisableLlwuClock (uint32_t *instance*) [inline], [static]

This function disables the clock for LLWU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.178 static bool CLOCK_SYS_GetLlwuGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for LLWU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

Function Documentation

**42.6.179 static void CLOCK_SYS_SetEnetExternalFreq (uint32_t *srcInstance*,
 uint32_t *freq*) [inline], [static]**

This function sets the ENET external clock frequency (ENET_1588_CLKIN).

Parameters

<i>srcInstance</i>	External source instance.
<i>freq</i>	Frequency value.

42.6.180 `uint32_t CLOCK_SYS_GetDmaFreq (uint32_t instance)`

This function gets the clock frequency for DMA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for DMA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.181 `uint32_t CLOCK_SYS_GetDmamuxFreq (uint32_t instance)`

This function gets the clock frequency for DMAMUX module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for DMAMUX module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.182 uint32_t CLOCK_SYS_GetPortFreq (uint32_t *instance*)

This function gets the clock frequency for PORT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for PORT module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.183 uint32_t CLOCK_SYS_GetMpuFreq (uint32_t *instance*)

This function gets the clock frequency for MPU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for MPU module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.184 uint32_t CLOCK_SYS_GetFlexbusFreq (uint32_t *instance*)

This function gets the clock frequency for FLEXBUS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for FLEXBUS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.185 uint32_t CLOCK_SYS_GetRngaFreq (uint32_t *instance*)

This function gets the clock frequency for RNGA module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for RNGA module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.186 uint32_t CLOCK_SYS_GetAdcFreq (uint32_t *instance*)

This function gets the clock frequency for ADC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for ADC module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.187 uint32_t CLOCK_SYS_GetFtmFreq (uint32_t *instance*)

(FlexTimer)

This function gets the clock frequency for FTM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

(FlexTimer)

This function gets the clock frequency for FTM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.188 `uint32_t CLOCK_SYS_GetUsbFreq (uint32_t instance)`

This function gets the clock frequency for USB FS OTG module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for USB FS OTG module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.189 `uint32_t CLOCK_SYS_GetSaiFreq (uint32_t instance)`

This function gets the clock frequency for I2S module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the clock frequency for I2S module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.190 static void CLOCK_SYS_EnableUsbClock (uint32_t *instance*) [inline], [static]

This function enables the clock for USBFS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.191 static void CLOCK_SYS_DisableUsbClock (uint32_t *instance*) [inline], [static]

This function disables the clock for USBFS module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.192 static bool CLOCK_SYS_GetUsbGateCmd (uint32_t *instance*) [inline], [static]

This function will get the clock gate state for USB module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.193 uint32_t CLOCK_SYS_GetTpmFreq (uint32_t *instance*)

Gets TPM clock frequency.

This function gets the clock frequency for TPM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

This function gets the TPM clock frequency.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

Current frequency.

42.6.194 static void CLOCK_SYS_SetTpmSrc (uint32_t *instance*, clock_tpm_src_t *tpmSrc*) [inline], [static]

This function sets the TPM clock source selection.

Parameters

<i>instance</i>	IP instance.
<i>tpmSrc</i>	The value to set.

42.6.195 static clock_tpm_src_t CLOCK_SYS_GetTpmSrc (uint32_t *instance*) [inline], [static]

This function gets the TPM clock source selection.

Function Documentation

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

Current selection.

42.6.196 uint32_t CLOCK_SYS_GetTpmExternalFreq (uint32_t *instance*)

This function gets the TPM external clock source frequency.

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

TPM external clock frequency.

42.6.197 static void CLOCK_SYS_SetTpmExternalSrc (uint32_t *instance*, sim_tpm_clk_sel_t *src*) [inline], [static]

This function sets the TPM external clock source selection.

Parameters

<i>instance</i>	IP instance.
<i>src</i>	TPM external clock source.

42.6.198 static sim_tpm_clk_sel_t CLOCK_SYS_GetTpmExternalSrc (uint32_t *instance*) [inline], [static]

This function sets the TPM external clock source selection.

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

setting Current TPM external clock source.

42.6.199 void CLOCK_SYS_EnableTpmClock (uint32_t *instance*) [inline]

This function enables the clock for TPM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.200 void CLOCK_SYS_DisableTpmClock (uint32_t *instance*) [inline]

This function disables the clock for TPM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.201 bool CLOCK_SYS_GetTpmGateCmd (uint32_t *instance*) [inline]

This function gets the clock gate state for TPM module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

This function will get the clock gate state for TPM module.

Function Documentation

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.202 static void CLOCK_SYS_SetTpmExternalFreq (*uint32_t srcInstance*, *uint32_t freq*) [inline], [static]

This function sets the TPM external clock frequency (TPM_CLKx).

Parameters

<i>srcInstance</i>	External source instance.
<i>freq</i>	Frequency value.

42.6.203 uint32_t CLOCK_SYS_GetCopFreq (*uint32_t instance*, *clock_cop_src_t copSrc*)

This function gets the COP clock frequency.

Parameters

<i>instance</i>	module device instance
<i>copSrc</i>	COP clock source selection.

Returns

Current frequency.

42.6.204 uint32_t CLOCK_SYS_GetRtcFreq (*uint32_t instance*, *clock_er32k_src_t rtcSrc*)

This function gets the rtc clock frequency.

Parameters

<i>instance</i>	module device instance
<i>rtcSrc</i>	rtc clock source selection.

Returns

Current frequency.

42.6.205 `uint32_t CLOCK_SYS_GetLpisciFreq (uint32_t instance)`

This function gets the clock frequency for LPSCI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

freq clock frequency for this module

42.6.206 `static void CLOCK_SYS_SetLpisciSrc (uint32_t instance, clock_lpisci_src_t lpisciSrc) [inline], [static]`

This function sets the LPSCI clock source selection.

Parameters

<i>instance</i>	IP instance.
<i>lpisciSrc</i>	The value to set.

42.6.207 `static clock_lpisci_src_t CLOCK_SYS_GetLpisciSrc (uint32_t instance) [inline], [static]`

This function gets the LPSCI clock source selection.

Function Documentation

Parameters

<i>instance</i>	IP instance.
-----------------	--------------

Returns

Current selection.

42.6.208 void CLOCK_SYS_EnableLpisciClock (uint32_t *instance*)

This function enables the clock for LPSCI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.209 void CLOCK_SYS_DisableLpisciClock (uint32_t *instance*)

This function disables the clock for LPSCI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

42.6.210 bool CLOCK_SYS_GetLpisciGateCmd (uint32_t *instance*)

This function will get the clock gate state for LPSCI module.

Parameters

<i>instance</i>	module device instance
-----------------	------------------------

Returns

state true - ungated(Enabled), false - gated (Disabled)

42.6.211 static void CLOCK_SYS_SetOutDiv5 (uint8_t *outdiv5*) [inline], [static]

This function sets divide value OUTDIV5.

Parameters

<i>outdiv5</i>	Outdivider5 setting
----------------	---------------------

42.6.212 static uint8_t CLOCK_SYS_GetOutDiv5(void) [inline], [static]

This function gets divide value OUTDIV5.

Returns

Outdivider5 setting

42.6.213 uint32_t CLOCK_SYS_GetOutdiv5ClockFreq(void)

This function gets the OUTDIV5 output clock for ADC.

Returns

freq Current frequency.

42.7 Variable Documentation

42.7.1 const uint32_t g_simBaseAddr[]

42.7.2 const uint32_t g_mcgBaseAddr[]

42.7.3 const uint32_t g_oscBaseAddr[]

42.7.4 clock_manager_user_config_t g_defaultClockConfigurations[]

K02F12810

42.8 K02F12810

The data structure definition for K02F12810 clock manager.

42.8.1 Overview

Files

- file [fsl_clock_MK02F12810.h](#)

Data Structures

- struct [osc_user_config_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k02f12810_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k02f12810_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency.

42.8.2 Data Structure Documentation

42.8.2.1 struct osc_user_config_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.8.2.1.0.6 Field Documentation

42.8.2.1.0.6.1 bool osc_user_config_t::enableExternalRef

42.8.2.1.0.6.2 bool osc_user_config_t::enableExternalRefInStop

42.8.2.1.0.6.3 mcg_high_gain_osc_select_t osc_user_config_t::hgo

42.8.2.1.0.6.4 mcg_external_ref_clock_select_t osc_user_config_t::erefs

42.8.2.1.0.6.5 uint32_t osc_user_config_t::freq

42.8.2.2 struct oscer_config_k02f12810_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.8.2.2.0.7 Field Documentation

42.8.2.2.0.7.1 bool oscer_config_k02f12810_t::Enable

42.8.2.2.0.7.2 bool oscer_config_k02f12810_t::EnableInStop

42.8.2.3 struct sim_config_k02f12810_t

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.8.2.3.0.8 Field Documentation

42.8.2.3.0.8.1 `clock_pllifl_sel_t sim_config_k02f12810_t::PlIflISel`

42.8.2.3.0.8.2 `clock_er32k_src_t sim_config_k02f12810_t::er32kSrc`

42.8.2.3.0.8.3 `uint8_t sim_config_k02f12810_t::outdiv4`

42.8.3 Macro Definition Documentation

42.8.3.1 `#define FTM_EXT_CLK_COUNT 2`

42.8.3.2 `#define CLOCK_CONFIG_NUM 3`

42.8.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.8.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.8.3.5 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.8.4 Variable Documentation

42.8.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.9 K22F12810

The data structure definition for K22F12810 clock manager.

42.9.1 Overview

Files

- file [fsl_clock_MK22F12810.h](#)

Data Structures

- struct [osc_user_config_k22f12810_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k22f12810_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k22f12810_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency.
- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency.

42.9.2 Data Structure Documentation

42.9.2.1 struct osc_user_config_k22f12810_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.9.2.1.0.9 Field Documentation

42.9.2.1.0.9.1 `bool osc_user_config_k22f12810_t::enableExternalRef`

42.9.2.1.0.9.2 `bool osc_user_config_k22f12810_t::enableExternalRefInStop`

42.9.2.1.0.9.3 `mcg_high_gain_osc_select_t osc_user_config_k22f12810_t::hgo`

42.9.2.1.0.9.4 `mcg_external_ref_clock_select_t osc_user_config_k22f12810_t::erefs`

42.9.2.1.0.9.5 `uint32_t osc_user_config_k22f12810_t::freq`

42.9.2.2 struct oscer_config_k22f12810_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.9.2.2.0.10 Field Documentation

42.9.2.2.0.10.1 `bool oscer_config_k22f12810_t::Enable`

42.9.2.2.0.10.2 `bool oscer_config_k22f12810_t::EnableInStop`

42.9.2.3 `struct sim_config_k22f12810_t`

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.9.2.3.0.11 Field Documentation

42.9.2.3.0.11.1 `clock_pllfl_sel_t sim_config_k22f12810_t::PllFlSel`

42.9.2.3.0.11.2 `clock_er32k_src_t sim_config_k22f12810_t::er32kSrc`

42.9.2.3.0.11.3 `uint8_t sim_config_k22f12810_t::outdiv4`

42.9.3 Macro Definition Documentation

42.9.3.1 `#define USB_EXT_CLK_COUNT 1`

42.9.3.2 `#define FTM_EXT_CLK_COUNT 2`

42.9.3.3 `#define CLOCK_CONFIG_NUM 3`

42.9.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.9.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.9.3.6 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.9.4 Variable Documentation

42.9.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

42.9.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

K22F25612

42.10 K22F25612

The data structure definition for K22F25612 clock manager.

42.10.1 Overview

Files

- file [fsl_clock_MK22F25612.h](#)

Data Structures

- struct [osc_user_config_k22f25612_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k22f25612_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k22f25612_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency.
- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency.

42.10.2 Data Structure Documentation

42.10.2.1 struct osc_user_config_k22f25612_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.10.2.1.0.12 Field Documentation

42.10.2.1.0.12.1 bool osc_user_config_k22f25612_t::enableExternalRef

42.10.2.1.0.12.2 bool osc_user_config_k22f25612_t::enableExternalRefInStop

42.10.2.1.0.12.3 mcg_high_gain_osc_select_t osc_user_config_k22f25612_t::hgo

42.10.2.1.0.12.4 mcg_external_ref_clock_select_t osc_user_config_k22f25612_t::erefs

42.10.2.1.0.12.5 uint32_t osc_user_config_k22f25612_t::freq

42.10.2.2 struct oscer_config_k22f25612_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

K22F25612

42.10.2.2.0.13 Field Documentation

42.10.2.2.0.13.1 `bool oscer_config_k22f25612_t::Enable`

42.10.2.2.0.13.2 `bool oscer_config_k22f25612_t::EnableInStop`

42.10.2.3 `struct sim_config_k22f25612_t`

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.10.2.3.0.14 Field Documentation

42.10.2.3.0.14.1 `clock_pllfl_sel_t sim_config_k22f25612_t::PllFlSel`

42.10.2.3.0.14.2 `clock_er32k_src_t sim_config_k22f25612_t::er32kSrc`

42.10.2.3.0.14.3 `uint8_t sim_config_k22f25612_t::outdiv4`

42.10.3 Macro Definition Documentation

42.10.3.1 `#define USB_EXT_CLK_COUNT 1`

42.10.3.2 `#define FTM_EXT_CLK_COUNT 2`

42.10.3.3 `#define CLOCK_CONFIG_NUM 3`

42.10.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.10.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.10.3.6 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.10.4 Variable Documentation

42.10.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

42.10.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.11 K22F51212

The data structure definition for K22F51212 clock manager.

42.11.1 Overview

Files

- file [fsl_clock_MK22F51212.h](#)

Data Structures

- struct [osc_user_config_k22f51212_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k22f51212_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k22f51212_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency.
- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency.

42.11.2 Data Structure Documentation

42.11.2.1 struct osc_user_config_k22f51212_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.11.2.1.0.15 Field Documentation

42.11.2.1.0.15.1 `bool osc_user_config_k22f51212_t::enableExternalRef`

42.11.2.1.0.15.2 `bool osc_user_config_k22f51212_t::enableExternalRefInStop`

42.11.2.1.0.15.3 `mcg_high_gain_osc_select_t osc_user_config_k22f51212_t::hgo`

42.11.2.1.0.15.4 `mcg_external_ref_clock_select_t osc_user_config_k22f51212_t::erefs`

42.11.2.1.0.15.5 `uint32_t osc_user_config_k22f51212_t::freq`

42.11.2.2 struct oscer_config_k22f51212_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.11.2.2.0.16 Field Documentation

42.11.2.2.0.16.1 `bool oscer_config_k22f51212_t::Enable`

42.11.2.2.0.16.2 `bool oscer_config_k22f51212_t::EnableInStop`

42.11.2.3 `struct sim_config_k22f51212_t`

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.11.2.3.0.17 Field Documentation

42.11.2.3.0.17.1 `clock_pllfl_sel_t sim_config_k22f51212_t::PllFlSel`

42.11.2.3.0.17.2 `clock_er32k_src_t sim_config_k22f51212_t::er32kSrc`

42.11.2.3.0.17.3 `uint8_t sim_config_k22f51212_t::outdiv4`

42.11.3 Macro Definition Documentation

42.11.3.1 `#define USB_EXT_CLK_COUNT 1`

42.11.3.2 `#define FTM_EXT_CLK_COUNT 2`

42.11.3.3 `#define CLOCK_CONFIG_NUM 3`

42.11.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.11.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.11.3.6 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.11.4 Variable Documentation

42.11.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

42.11.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.12 K24F12

The data structure definition for K24F12 clock manager.

42.12.1 Overview

Files

- file [fsl_clock_MK24F12.h](#)

Data Structures

- struct [osc_user_config_k24f12_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k24f12_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k24f12_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [SDHC_EXT_CLK_COUNT](#) 1
SDHC external clock source count.
- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 2
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.

Variables

- uint32_t [g_sdhcClkInFreq](#) [[SDHC_EXT_CLK_COUNT](#)]
SDHC external clock frequency(SDHC_CLKIN).
- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency(USB_CLKIN).
- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.12.2 Data Structure Documentation

42.12.2.1 struct osc_user_config_k24f12_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.12.2.1.0.18 Field Documentation

42.12.2.1.0.18.1 `bool osc_user_config_k24f12_t::enableExternalRef`

42.12.2.1.0.18.2 `bool osc_user_config_k24f12_t::enableExternalRefInStop`

42.12.2.1.0.18.3 `mcg_high_gain_osc_select_t osc_user_config_k24f12_t::hgo`

42.12.2.1.0.18.4 `mcg_external_ref_clock_select_t osc_user_config_k24f12_t::erefs`

42.12.2.1.0.18.5 `uint32_t osc_user_config_k24f12_t::freq`

42.12.2.2 struct oscer_config_k24f12_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

K24F12

42.12.2.2.0.19 Field Documentation

42.12.2.2.0.19.1 `bool oscer_config_k24f12_t::Enable`

42.12.2.2.0.19.2 `bool oscer_config_k24f12_t::EnableInStop`

42.12.2.3 `struct sim_config_k24f12_t`

Data Fields

- `clock_pllfll_sel_t PllFllSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.12.2.3.0.20 Field Documentation

42.12.2.3.0.20.1 `clock_pllfll_sel_t sim_config_k24f12_t::PllFllSel`

42.12.2.3.0.20.2 `clock_er32k_src_t sim_config_k24f12_t::er32kSrc`

42.12.2.3.0.20.3 `uint8_t sim_config_k24f12_t::outdiv4`

42.12.3 Macro Definition Documentation

42.12.3.1 `#define SDHC_EXT_CLK_COUNT 1`

42.12.3.2 `#define USB_EXT_CLK_COUNT 1`

42.12.3.3 `#define FTM_EXT_CLK_COUNT 2`

42.12.3.4 `#define CLOCK_CONFIG_NUM 2`

42.12.3.5 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.12.3.6 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.12.4 Variable Documentation

42.12.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

42.12.4.2 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

42.12.4.3 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.13 K24F25612

The data structure definition for K24F25612 clock manager.

42.13.1 Overview

Files

- file [fsl_clock_MK24F25612.h](#)

Data Structures

- struct [osc_user_config_k24f25612_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k24f25612_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k24f25612_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 2
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.

Variables

- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency(USB_CLKIN).
- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.13.2 Data Structure Documentation

42.13.2.1 struct osc_user_config_k24f25612_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.13.2.1.0.21 Field Documentation

42.13.2.1.0.21.1 `bool osc_user_config_k24f25612_t::enableExternalRef`

42.13.2.1.0.21.2 `bool osc_user_config_k24f25612_t::enableExternalRefInStop`

42.13.2.1.0.21.3 `mcg_high_gain_osc_select_t osc_user_config_k24f25612_t::hgo`

42.13.2.1.0.21.4 `mcg_external_ref_clock_select_t osc_user_config_k24f25612_t::erefs`

42.13.2.1.0.21.5 `uint32_t osc_user_config_k24f25612_t::freq`

42.13.2.2 struct oscer_config_k24f25612_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.13.2.2.0.22 Field Documentation

42.13.2.2.0.22.1 `bool oscer_config_k24f25612_t::Enable`

42.13.2.2.0.22.2 `bool oscer_config_k24f25612_t::EnableInStop`

42.13.2.3 struct sim_config_k24f25612_t

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.13.2.3.0.23 Field Documentation

42.13.2.3.0.23.1 `clock_pllfill_sel_t sim_config_k24f25612_t::PllFillSel`

42.13.2.3.0.23.2 `clock_er32k_src_t sim_config_k24f25612_t::er32kSrc`

42.13.2.3.0.23.3 `uint8_t sim_config_k24f25612_t::outdiv4`

42.13.3 Macro Definition Documentation

42.13.3.1 `#define USB_EXT_CLK_COUNT 1`

42.13.3.2 `#define FTM_EXT_CLK_COUNT 2`

42.13.3.3 `#define CLOCK_CONFIG_NUM 2`

42.13.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.13.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.13.4 Variable Documentation

42.13.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

42.13.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

K60D10

42.14 K60D10

The data structure definition for K60D10 clock manager.

42.14.1 Overview

Files

- file `fsl_clock_MK60D10.h`

Data Structures

- struct `osc_user_config_k60d10_t`
OSC Initialization Configuration Structure. [More...](#)
- struct `oscer_config_k60d10_t`
OSC configuration for OSCERCLK. [More...](#)
- struct `sim_config_k60d10_t`
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- `#define ENET_EXT_CLK_COUNT 1`
ENET external clock source count.
- `#define SDHC_EXT_CLK_COUNT 1`
SDHC external clock source count.
- `#define USB_EXT_CLK_COUNT 1`
USB external clock source count.
- `#define FTM_EXT_CLK_COUNT 2`
FTM external clock source count.
- `#define CLOCK_CONFIG_NUM 2`
Default clock configuration number.
- `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`
Clock configuration index for VLPR mode.
- `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`
Clock configuration index for RUN mode.

Variables

- `uint32_t g_enet1588ClkInFreq [ENET_EXT_CLK_COUNT]`
ENET external clock frequency(ENET_1588_CLKIN).
- `uint32_t g_sdhcClkInFreq [SDHC_EXT_CLK_COUNT]`
SDHC external clock frequency(SDHC_CLKIN).
- `uint32_t g_usbClkInFreq [USB_EXT_CLK_COUNT]`
USB external clock frequency(USB_CLKIN).
- `uint32_t g_ftmClkFreq [FTM_EXT_CLK_COUNT]`
FTM external clock frequency(FTM_CLK).

42.14.2 Data Structure Documentation

42.14.2.1 struct osc_user_config_k60d10_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.14.2.1.0.24 Field Documentation

42.14.2.1.0.24.1 bool osc_user_config_k60d10_t::enableExternalRef

42.14.2.1.0.24.2 bool osc_user_config_k60d10_t::enableExternalRefInStop

42.14.2.1.0.24.3 mcg_high_gain_osc_select_t osc_user_config_k60d10_t::hgo

42.14.2.1.0.24.4 mcg_external_ref_clock_select_t osc_user_config_k60d10_t::erefs

42.14.2.1.0.24.5 uint32_t osc_user_config_k60d10_t::freq

42.14.2.2 struct oscer_config_k60d10_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

K60D10

42.14.2.2.0.25 Field Documentation

42.14.2.2.0.25.1 `bool oscer_config_k60d10_t::Enable`

42.14.2.2.0.25.2 `bool oscer_config_k60d10_t::EnableInStop`

42.14.2.3 `struct sim_config_k60d10_t`

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.14.2.3.0.26 Field Documentation

- 42.14.2.3.0.26.1 `clock_pllfill_sel_t sim_config_k60d10_t::PlIIFIISel`
- 42.14.2.3.0.26.2 `clock_er32k_src_t sim_config_k60d10_t::er32kSrc`
- 42.14.2.3.0.26.3 `uint8_t sim_config_k60d10_t::outdiv4`

42.14.3 Macro Definition Documentation

- 42.14.3.1 `#define ENET_EXT_CLK_COUNT 1`
- 42.14.3.2 `#define SDHC_EXT_CLK_COUNT 1`
- 42.14.3.3 `#define USB_EXT_CLK_COUNT 1`
- 42.14.3.4 `#define FTM_EXT_CLK_COUNT 2`
- 42.14.3.5 `#define CLOCK_CONFIG_NUM 2`
- 42.14.3.6 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`
- 42.14.3.7 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.14.4 Variable Documentation

- 42.14.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`
- 42.14.4.2 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`
- 42.14.4.3 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

K64F12

42.15 K64F12

The data structure definition for K64F12 clock manager.

42.15.1 Overview

Files

- file [fsl_clock_MK64F12.h](#)

Data Structures

- struct [osc_user_config_k64f12_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_k64f12_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_k64f12_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [ENET_EXT_CLK_COUNT](#) 1
ENET external clock source count.
- #define [SDHC_EXT_CLK_COUNT](#) 1
SDHC external clock source count.
- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 2
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.

Variables

- uint32_t [g_enet1588ClkInFreq](#) [[ENET_EXT_CLK_COUNT](#)]
ENET external clock frequency(ENET_1588_CLKIN).
- uint32_t [g_sdhcClkInFreq](#) [[SDHC_EXT_CLK_COUNT](#)]
SDHC external clock frequency(SDHC_CLKIN).
- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency(USB_CLKIN).
- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.15.2 Data Structure Documentation

42.15.2.1 struct osc_user_config_k64f12_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.15.2.1.0.27 Field Documentation

42.15.2.1.0.27.1 `bool osc_user_config_k64f12_t::enableExternalRef`

42.15.2.1.0.27.2 `bool osc_user_config_k64f12_t::enableExternalRefInStop`

42.15.2.1.0.27.3 `mcg_high_gain_osc_select_t osc_user_config_k64f12_t::hgo`

42.15.2.1.0.27.4 `mcg_external_ref_clock_select_t osc_user_config_k64f12_t::erefs`

42.15.2.1.0.27.5 `uint32_t osc_user_config_k64f12_t::freq`

42.15.2.2 struct oscer_config_k64f12_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.15.2.2.0.28 Field Documentation

42.15.2.2.0.28.1 `bool oscer_config_k64f12_t::Enable`

42.15.2.2.0.28.2 `bool oscer_config_k64f12_t::EnableInStop`

42.15.2.3 `struct sim_config_k64f12_t`

Data Fields

- `clock_pllfll_sel_t PllFllSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.15.2.3.0.29 Field Documentation

42.15.2.3.0.29.1 `clock_pllfill_sel_t sim_config_k64f12_t::PlIIFIISel`

42.15.2.3.0.29.2 `clock_er32k_src_t sim_config_k64f12_t::er32kSrc`

42.15.2.3.0.29.3 `uint8_t sim_config_k64f12_t::outdiv4`

42.15.3 Macro Definition Documentation

42.15.3.1 `#define ENET_EXT_CLK_COUNT 1`

42.15.3.2 `#define SDHC_EXT_CLK_COUNT 1`

42.15.3.3 `#define USB_EXT_CLK_COUNT 1`

42.15.3.4 `#define FTM_EXT_CLK_COUNT 2`

42.15.3.5 `#define CLOCK_CONFIG_NUM 2`

42.15.3.6 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.15.3.7 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.15.4 Variable Documentation

42.15.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

42.15.4.2 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

42.15.4.3 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

KL03Z4

42.16 KL03Z4

The data structure definition for KL03Z4 clock manager.

42.16.1 Overview

Files

- file [fsl_clock_MKL03Z4.h](#)

Data Structures

- struct [oscer_config_kl03z4_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kl03z4_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [TPM_EXT_CLK_COUNT](#) 2
TPM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 2
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.

Variables

- uint32_t [g_tpmClkFreq](#) [[TPM_EXT_CLK_COUNT](#)]
TPM external clock frequency(TPM_CLK).

42.16.2 Data Structure Documentation

42.16.2.1 struct oscer_config_kl03z4_t

Data Fields

- bool [Enable](#)
OSCERCLK enable or not.
- bool [EnableInStop](#)
OSCERCLK enable or not in stop mode.

42.16.2.1.0.30 Field Documentation

42.16.2.1.0.30.1 `bool oscer_config_kl03z4_t::Enable`

42.16.2.1.0.30.2 `bool oscer_config_kl03z4_t::EnableInStop`

42.16.2.2 `struct sim_config_kl03z4_t`

Data Fields

- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.16.2.2.0.31 Field Documentation

42.16.2.2.0.31.1 `clock_er32k_src_t sim_config_kl03z4_t::er32kSrc`

42.16.2.2.0.31.2 `uint8_t sim_config_kl03z4_t::outdiv4`

42.16.3 Macro Definition Documentation

42.16.3.1 `#define TPM_EXT_CLK_COUNT 2`

42.16.3.2 `#define CLOCK_CONFIG_NUM 2`

42.16.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.16.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.16.4 Variable Documentation

42.16.4.1 `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

42.17 KL46Z4

The data structure definition for KL46Z4 clock manager.

42.17.1 Overview

Files

- file [fsl_clock_MKL46Z4.h](#)

Data Structures

- struct [osc_user_config_kl46z4_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_kl46z4_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kl46z4_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [TPM_EXT_CLK_COUNT](#) 2
TPM external clock source count.
- #define [USB_EXT_CLK_COUNT](#) 1
USB external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 2
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.

Variables

- uint32_t [g_tpmClkFreq](#) [[TPM_EXT_CLK_COUNT](#)]
TPM external clock frequency(TPM_CLK).
- uint32_t [g_usbClkInFreq](#) [[USB_EXT_CLK_COUNT](#)]
USB external clock frequency(USB_CLKIN).

42.17.2 Data Structure Documentation

42.17.2.1 struct osc_user_config_kl46z4_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.17.2.1.0.32 Field Documentation

42.17.2.1.0.32.1 `bool osc_user_config_kl46z4_t::enableExternalRef`

42.17.2.1.0.32.2 `bool osc_user_config_kl46z4_t::enableExternalRefInStop`

42.17.2.1.0.32.3 `mcg_high_gain_osc_select_t osc_user_config_kl46z4_t::hgo`

42.17.2.1.0.32.4 `mcg_external_ref_clock_select_t osc_user_config_kl46z4_t::erefs`

42.17.2.1.0.32.5 `uint32_t osc_user_config_kl46z4_t::freq`

42.17.2.2 struct oscer_config_kl46z4_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.17.2.2.0.33 Field Documentation

42.17.2.2.0.33.1 `bool oscer_config_kl46z4_t::Enable`

42.17.2.2.0.33.2 `bool oscer_config_kl46z4_t::EnableInStop`

42.17.2.3 struct sim_config_kl46z4_t

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.17.2.3.0.34 Field Documentation**42.17.2.3.0.34.1 clock_pllfill_sel_t sim_config_kl46z4_t::PllFillSel****42.17.2.3.0.34.2 clock_er32k_src_t sim_config_kl46z4_t::er32kSrc****42.17.2.3.0.34.3 uint8_t sim_config_kl46z4_t::outdiv4****42.17.3 Macro Definition Documentation****42.17.3.1 #define TPM_EXT_CLK_COUNT 2****42.17.3.2 #define USB_EXT_CLK_COUNT 1****42.17.3.3 #define CLOCK_CONFIG_NUM 2****42.17.3.4 #define CLOCK_CONFIG_INDEX_FOR_VLPR 0****42.17.3.5 #define CLOCK_CONFIG_INDEX_FOR_RUN 1****42.17.4 Variable Documentation****42.17.4.1 uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]****42.17.4.2 uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]**

42.18 KV10Z7

The data structure definition for KV10Z7 clock manager.

42.18.1 Overview

Files

- file [fsl_clock_MKV10Z7.h](#)

Data Structures

- struct [osc_user_config_kv10z7_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_kv10z7_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kv10z7_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [FTM_EXT_CLK_COUNT](#) 3
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 2
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.

Variables

- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.18.2 Data Structure Documentation

42.18.2.1 struct osc_user_config_kv10z7_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.18.2.1.0.35 Field Documentation

42.18.2.1.0.35.1 `bool osc_user_config_kv10z7_t::enableExternalRef`

42.18.2.1.0.35.2 `bool osc_user_config_kv10z7_t::enableExternalRefInStop`

42.18.2.1.0.35.3 `mcg_high_gain_osc_select_t osc_user_config_kv10z7_t::hgo`

42.18.2.1.0.35.4 `mcg_external_ref_clock_select_t osc_user_config_kv10z7_t::erefs`

42.18.2.1.0.35.5 `uint32_t osc_user_config_kv10z7_t::freq`

42.18.2.2 struct oscer_config_kv10z7_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.18.2.2.0.36 Field Documentation

42.18.2.2.0.36.1 `bool oscer_config_kv10z7_t::Enable`

42.18.2.2.0.36.2 `bool oscer_config_kv10z7_t::EnableInStop`

42.18.2.3 struct sim_config_kv10z7_t

Data Fields

- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv5`
OUTDIV setting.
- bool `outdiv5Enable`
OUTDIV5EN.

42.18.2.3.0.37 Field Documentation

42.18.2.3.0.37.1 `clock_er32k_src_t sim_config_kv10z7_t::er32kSrc`

42.18.2.3.0.37.2 `uint8_t sim_config_kv10z7_t::outdiv5`

42.18.2.3.0.37.3 `bool sim_config_kv10z7_t::outdiv5Enable`

42.18.3 Macro Definition Documentation

42.18.3.1 `#define FTM_EXT_CLK_COUNT 3`

42.18.3.2 `#define CLOCK_CONFIG_NUM 2`

42.18.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.18.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.18.4 Variable Documentation

42.18.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.19 KV30F12810

The data structure definition for KV30F12810 clock manager.

42.19.1 Overview

Files

- file [fsl_clock_MKV30F12810.h](#)

Data Structures

- struct [osc_user_config_kv30f12810_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_kv30f12810_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kv30f12810_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.19.2 Data Structure Documentation

42.19.2.1 struct osc_user_config_kv30f12810_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.19.2.1.0.38 Field Documentation

42.19.2.1.0.38.1 `bool osc_user_config_kv30f12810_t::enableExternalRef`

42.19.2.1.0.38.2 `bool osc_user_config_kv30f12810_t::enableExternalRefInStop`

42.19.2.1.0.38.3 `mcg_high_gain_osc_select_t osc_user_config_kv30f12810_t::hgo`

42.19.2.1.0.38.4 `mcg_external_ref_clock_select_t osc_user_config_kv30f12810_t::erefs`

42.19.2.1.0.38.5 `uint32_t osc_user_config_kv30f12810_t::freq`

42.19.2.2 struct oscer_config_kv30f12810_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.19.2.2.0.39 Field Documentation

42.19.2.2.0.39.1 `bool oscer_config_kv30f12810_t::Enable`

42.19.2.2.0.39.2 `bool oscer_config_kv30f12810_t::EnableInStop`

42.19.2.3 struct sim_config_kv30f12810_t

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.19.2.3.0.40 Field Documentation

42.19.2.3.0.40.1 `clock_pllfill_sel_t sim_config_kv30f12810_t::PlIIFIISel`

42.19.2.3.0.40.2 `clock_er32k_src_t sim_config_kv30f12810_t::er32kSrc`

42.19.2.3.0.40.3 `uint8_t sim_config_kv30f12810_t::outdiv4`

42.19.3 Macro Definition Documentation

42.19.3.1 `#define FTM_EXT_CLK_COUNT 2`

42.19.3.2 `#define CLOCK_CONFIG_NUM 3`

42.19.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.19.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.19.3.5 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.19.4 Variable Documentation

42.19.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.20 KV31F12810

The data structure definition for KV31F12810 clock manager.

42.20.1 Overview

Files

- file [fsl_clock_MKV31F12810.h](#)

Data Structures

- struct [osc_user_config_kv31f12810_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_kv31f12810_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kv31f12810_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.20.2 Data Structure Documentation

42.20.2.1 struct osc_user_config_kv31f12810_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool [enableExternalRef](#)
Enable OSCERCLK.
- bool [enableExternalRefInStop](#)
Enable OSCERCLK in stop mode.
- [mcg_high_gain_osc_select_t hgo](#)
High gain oscillator select.
- [mcg_external_ref_clock_select_t erefs](#)
External reference select.
- uint32_t [freq](#)
External clock frequency.

42.20.2.1.0.41 Field Documentation

42.20.2.1.0.41.1 [bool osc_user_config_kv31f12810_t::enableExternalRef](#)

42.20.2.1.0.41.2 [bool osc_user_config_kv31f12810_t::enableExternalRefInStop](#)

42.20.2.1.0.41.3 [mcg_high_gain_osc_select_t osc_user_config_kv31f12810_t::hgo](#)

42.20.2.1.0.41.4 [mcg_external_ref_clock_select_t osc_user_config_kv31f12810_t::erefs](#)

42.20.2.1.0.41.5 [uint32_t osc_user_config_kv31f12810_t::freq](#)

42.20.2.2 struct oscer_config_kv31f12810_t

Data Fields

- bool [Enable](#)
OSCERCLK enable or not.
- bool [EnableInStop](#)
OSCERCLK enable or not in stop mode.

42.20.2.2.0.42 Field Documentation

42.20.2.2.0.42.1 [bool oscer_config_kv31f12810_t::Enable](#)

42.20.2.2.0.42.2 [bool oscer_config_kv31f12810_t::EnableInStop](#)

42.20.2.3 struct sim_config_kv31f12810_t

Data Fields

- [clock_pllfl_sel_t PllFlSel](#)
PLL/FLL/IRC48M selection.
- [clock_er32k_src_t er32kSrc](#)
ERCLK32K source selection.
- uint8_t [outdiv4](#)
OUTDIV setting.

42.20.2.3.0.43 Field Documentation

42.20.2.3.0.43.1 `clock_pllfill_sel_t sim_config_kv31f12810_t::PlIFlISel`

42.20.2.3.0.43.2 `clock_er32k_src_t sim_config_kv31f12810_t::er32kSrc`

42.20.2.3.0.43.3 `uint8_t sim_config_kv31f12810_t::outdiv4`

42.20.3 Macro Definition Documentation

42.20.3.1 `#define FTM_EXT_CLK_COUNT 2`

42.20.3.2 `#define CLOCK_CONFIG_NUM 3`

42.20.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.20.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.20.3.5 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.20.4 Variable Documentation

42.20.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.21 KV31F25612

The data structure definition for KV31F25612 clock manager.

42.21.1 Overview

Files

- file [fsl_clock_MKV31F25612.h](#)

Data Structures

- struct [osc_user_config_kv31f25612_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_kv31f25612_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kv31f25612_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.21.2 Data Structure Documentation

42.21.2.1 struct osc_user_config_kv31f25612_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool `enableExternalRef`
Enable OSCERCLK.
- bool `enableExternalRefInStop`
Enable OSCERCLK in stop mode.
- `mcg_high_gain_osc_select_t hgo`
High gain oscillator select.
- `mcg_external_ref_clock_select_t erefs`
External reference select.
- `uint32_t freq`
External clock frequency.

42.21.2.1.0.44 Field Documentation

42.21.2.1.0.44.1 `bool osc_user_config_kv31f25612_t::enableExternalRef`

42.21.2.1.0.44.2 `bool osc_user_config_kv31f25612_t::enableExternalRefInStop`

42.21.2.1.0.44.3 `mcg_high_gain_osc_select_t osc_user_config_kv31f25612_t::hgo`

42.21.2.1.0.44.4 `mcg_external_ref_clock_select_t osc_user_config_kv31f25612_t::erefs`

42.21.2.1.0.44.5 `uint32_t osc_user_config_kv31f25612_t::freq`

42.21.2.2 struct oscer_config_kv31f25612_t

Data Fields

- bool `Enable`
OSCERCLK enable or not.
- bool `EnableInStop`
OSCERCLK enable or not in stop mode.

42.21.2.2.0.45 Field Documentation

42.21.2.2.0.45.1 `bool oscer_config_kv31f25612_t::Enable`

42.21.2.2.0.45.2 `bool oscer_config_kv31f25612_t::EnableInStop`

42.21.2.3 struct sim_config_kv31f25612_t

Data Fields

- `clock_pllfl_sel_t PllFlSel`
PLL/FLL/IRC48M selection.
- `clock_er32k_src_t er32kSrc`
ERCLK32K source selection.
- `uint8_t outdiv4`
OUTDIV setting.

42.21.2.3.0.46 Field Documentation

42.21.2.3.0.46.1 `clock_pllfill_sel_t sim_config_kv31f25612_t::PlIIFIISel`

42.21.2.3.0.46.2 `clock_er32k_src_t sim_config_kv31f25612_t::er32kSrc`

42.21.2.3.0.46.3 `uint8_t sim_config_kv31f25612_t::outdiv4`

42.21.3 Macro Definition Documentation

42.21.3.1 `#define FTM_EXT_CLK_COUNT 2`

42.21.3.2 `#define CLOCK_CONFIG_NUM 3`

42.21.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.21.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.21.3.5 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.21.4 Variable Documentation

42.21.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

42.22 KV31F51212

The data structure definition for KV31F51212 clock manager.

42.22.1 Overview

Files

- file [fsl_clock_MKV31F51212.h](#)

Data Structures

- struct [osc_user_config_kv31f51212_t](#)
OSC Initialization Configuration Structure. [More...](#)
- struct [oscer_config_kv31f51212_t](#)
OSC configuration for OSCERCLK. [More...](#)
- struct [sim_config_kv31f51212_t](#)
SIM configuration structure for dynamic clock setting. [More...](#)

Macros

- #define [FTM_EXT_CLK_COUNT](#) 2
FTM external clock source count.
- #define [CLOCK_CONFIG_NUM](#) 3
Default clock configuration number.
- #define [CLOCK_CONFIG_INDEX_FOR_VLPR](#) 0
Clock configuration index for VLPR mode.
- #define [CLOCK_CONFIG_INDEX_FOR_RUN](#) 1
Clock configuration index for RUN mode.
- #define [CLOCK_CONFIG_INDEX_FOR_HSRUN](#) 2
Clock configuration index for HSRUN mode.

Variables

- uint32_t [g_ftmClkFreq](#) [[FTM_EXT_CLK_COUNT](#)]
FTM external clock frequency(FTM_CLK).

42.22.2 Data Structure Documentation

42.22.2.1 struct osc_user_config_kv31f51212_t

Defines the configuration data structure to initialize the OSC.

Data Fields

- bool [enableExternalRef](#)
Enable OSCERCLK.
- bool [enableExternalRefInStop](#)
Enable OSCERCLK in stop mode.
- [mcg_high_gain_osc_select_t](#) [hgo](#)
High gain oscillator select.
- [mcg_external_ref_clock_select_t](#) [eref](#)
External reference select.
- uint32_t [freq](#)
External clock frequency.

42.22.2.1.0.47 Field Documentation

42.22.2.1.0.47.1 [bool osc_user_config_kv31f51212_t::enableExternalRef](#)

42.22.2.1.0.47.2 [bool osc_user_config_kv31f51212_t::enableExternalRefInStop](#)

42.22.2.1.0.47.3 [mcg_high_gain_osc_select_t osc_user_config_kv31f51212_t::hgo](#)

42.22.2.1.0.47.4 [mcg_external_ref_clock_select_t osc_user_config_kv31f51212_t::eref](#)

42.22.2.1.0.47.5 [uint32_t osc_user_config_kv31f51212_t::freq](#)

42.22.2 struct oscer_config_kv31f51212_t

Data Fields

- bool [Enable](#)
OSCERCLK enable or not.
- bool [EnableInStop](#)
OSCERCLK enable or not in stop mode.

42.22.2.2.0.48 Field Documentation

42.22.2.2.0.48.1 [bool oscer_config_kv31f51212_t::Enable](#)

42.22.2.2.0.48.2 [bool oscer_config_kv31f51212_t::EnableInStop](#)

42.22.2.3 struct sim_config_kv31f51212_t

Data Fields

- [clock_pllfl_sel_t](#) [PllFlSel](#)
PLL/FLL/IRC48M selection.
- [clock_er32k_src_t](#) [er32kSrc](#)
ERCLK32K source selection.
- uint8_t [outdiv4](#)
OUTDIV setting.

42.22.2.3.0.49 Field Documentation

42.22.2.3.0.49.1 `clock_pllfill_sel_t sim_config_kv31f51212_t::PlIIFIISel`

42.22.2.3.0.49.2 `clock_er32k_src_t sim_config_kv31f51212_t::er32kSrc`

42.22.2.3.0.49.3 `uint8_t sim_config_kv31f51212_t::outdiv4`

42.22.3 Macro Definition Documentation

42.22.3.1 `#define FTM_EXT_CLK_COUNT 2`

42.22.3.2 `#define CLOCK_CONFIG_NUM 3`

42.22.3.3 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

42.22.3.4 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

42.22.3.5 `#define CLOCK_CONFIG_INDEX_FOR_HSRUN 2`

42.22.4 Variable Documentation

42.22.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

Chapter 43

Hwtimer_driver

43.1 Overview

The Kinetis SDK provides the HwTimer driver for various timer modules.

Modules

- [HwTimer Driver](#)

HwTimer Driver

43.2 HwTimer Driver

The HwTimer driver provides common API for various timer modules.

43.2.1 HwTimer Overview

The driver consists of two layers:

- The hardware-specific lower layer contains implementation specific for a particular timer module. An application should not use this layer.
- The generic upper layer provides abstraction to call the appropriate lower layer functions while passing the appropriate context structure to them. This chapter describes the generic upper layer only.

Chapter 44

Interrupt Manager (Interrupt)

The Kinetis SDK Interrupt Manager provides a set of API/services to configure the Interrupt Controller (NVIC).

44.1 Overview

Files

- file [fsl_interrupt_manager.h](#)

Enumerations

- enum [interrupt_status_t](#)
interrupt status return codes.

[interrupt_manager](#) APIs

- void * [INT_SYS_InstallHandler](#) (IRQn_Type irqNumber, void(*handler)(void))
Installs an interrupt handler routine for a given IRQ number.
- static void [INT_SYS_EnableIRQ](#) (IRQn_Type irqNumber)
Enables an interrupt for a given IRQ number.
- static void [INT_SYS_DisableIRQ](#) (IRQn_Type irqNumber)
Disables an interrupt for a given IRQ number.
- void [INT_SYS_EnableIRQGlobal](#) (void)
Enables system interrupt.
- void [INT_SYS_DisableIRQGlobal](#) (void)
Disable system interrupt.

44.1.1 Interrupt Manager

Overview

The Interrupt Manager provides a set of APIs so that the application can enable or disable an interrupt for a specific device and also set/get interrupt status, priority and other features. Also it provides a way to update the vector table for a specific device interrupt handler.

Interrupt Names

Each chip has its own set of supported interrupt names defined in the chip-specific header file. For example, for K70, the header file is MK70F12.h or MK70F15.h as an IRQn_Type.

Example to set/update the vector table for the I2C0_IRQHandler interrupt handler:

Function Documentation

```
#include "interrupt/fsl_interrupt_manager.h"  
  
interrupt_register_handler(I2C0_IRQn, irq_handler_I2C0_IRQn);
```

Example to enable/disable an interrupt for the I2C0_IRQn

```
#include "interrupt/fsl_interrupt_manager.h"  
  
interrupt_enable(I2C0_IRQn);  
  
interrupt_disable(I2C0_IRQn);
```

44.2 Enumeration Type Documentation

44.2.1 enum interrupt_status_t

44.3 Function Documentation

44.3.1 void* INT_SYS_InstallHandler (IRQn_Type *irqNumber*, void(*)(void) *handler*)

This function lets the application register/replace the interrupt handler for a specified IRQ number. The IRQ number is different than the vector number. IRQ 0 starts from the vector 16 address. See a chip-specific reference manual for details and the startup_MKxxxx.s file for each chip family to find out the default interrupt handler for each device. This function converts the IRQ number to the vector number by adding 16 to it.

Parameters

<i>irqNumber</i>	IRQ number
<i>handler</i>	Interrupt handler routine address pointer

Returns

Whether the installation succeed or not

44.3.2 static void INT_SYS_EnableIRQ (IRQn_Type *irqNumber*) [inline], [static]

This function enables the individual interrupt for a specified IRQ number. It calls the system NVIC API to access the interrupt control register. The input IRQ number does not include the core interrupt, only the peripheral interrupt, from 0 to a maximum supported IRQ.

Parameters

<i>irqNumber</i>	IRQ number
------------------	------------

44.3.3 static void INT_SYS_DisableIRQ (IRQn_Type *irqNumber*) [inline], [static]

This function enables the individual interrupt for a specified IRQ number. It calls the system NVIC API to access the interrupt control register.

Parameters

<i>irqNumber</i>	IRQ number
------------------	------------

44.3.4 void INT_SYS_EnableIRQGlobal (void)

This function enables the global interrupt by calling the core API.

44.3.5 void INT_SYS_DisableIRQGlobal (void)

This function disables the global interrupt by calling the core API.

Function Documentation

Chapter 45

Power Manager (Power)

45.1 Overview

The Kinetis SDK Power Manager provides a set of API/services to configure the power-related IPs, such as SMC, PMC, RCM, etc.

Modules

- Power Manager driver

Data Structures

- struct `power_manager_user_config_t`
Power mode user configuration structure. [More...](#)
- struct `power_manager_notify_struct_t`
Power notification structure passed to registered callback function. [More...](#)
- struct `power_manager_callback_user_config_t`
callback configuration structure [More...](#)
- struct `power_manager_state_t`
Power manager internal state structure. [More...](#)

Typedefs

- typedef void `power_manager_callback_data_t`
Callback-specific data.
- typedef
`power_manager_error_code_t(* power_manager_callback_t)(power_manager_notify_struct_t *notify, power_manager_callback_data_t *dataPtr)`
Callback prototype.

Enumerations

- enum `power_manager_modes_t` {
 kPowerManagerRun,
 kPowerManagerVlpr,
 kPowerManagerWait,
 kPowerManagerVlpw,
 kPowerManagerStop,
 kPowerManagerVlps,
 kPowerManagerVlls1,
 kPowerManagerVlls3 }

Power modes enumeration.

Overview

- enum `power_manager_error_code_t` {
 `kPowerManagerSuccess`,
 `kPowerManagerError`,
 `kPowerManagerErrorOutOfRange`,
 `kPowerManagerErrorSwitch`,
 `kPowerManagerErrorNotificationBefore`,
 `kPowerManagerErrorNotificationAfter`,
 `kPowerManagerErrorClock` }
 Power manager success code and error codes.
- enum `power_manager_policy_t` {
 `kPowerManagerPolicyAgreement`,
 `kPowerManagerPolicyForcible` }
 Power manager policies.
- enum `power_manager_notify_t` {
 `kPowerManagerNotifyRecover` = 0x00U,
 `kPowerManagerNotifyBefore` = 0x01U,
 `kPowerManagerNotifyAfter` = 0x02U }
 The PM notification type.
- enum `power_manager_callback_type_t` {
 `kPowerManagerCallbackBefore` = 0x01U,
 `kPowerManagerCallbackAfter` = 0x02U,
 `kPowerManagerCallbackBeforeAfter` = 0x03U }
 The callback type, indicates what kinds of notification this callback handles.

Functions

- `power_manager_error_code_t POWER_SYS_Init` (`power_manager_user_config_t const **(powerConfigsPtr)[]`, `uint8_t configsNumber`, `power_manager_callback_user_config_t const **(callbacksPtr)[]`, `uint8_t callbacksNumber`)
 Power manager initialization for operation.
- `power_manager_error_code_t POWER_SYS_Deinit` (`void`)
 This function deinitializes the Power manager.
- `power_manager_error_code_t POWER_SYS_SetMode` (`uint8_t powerModeIndex`, `power_manager_policy_t policy`)
 This function configures the power mode.
- `power_manager_error_code_t POWER_SYS_GetLastMode` (`uint8_t *powerModeIndexPtr`)
 This function returns power mode set as the last one.
- `power_manager_error_code_t POWER_SYS_GetLastModeConfig` (`power_manager_user_config_t **powerModePtr`)
 This function returns user configuration structure of power mode set as the last one.
- `power_manager_modes_t POWER_SYS_GetCurrentMode` (`void`)
 This function returns currently running power mode.
- `uint8_t POWER_SYS_GetErrorHandlerIndex` (`void`)
 This function returns the last failed notification callback.
- `power_manager_callback_user_config_t * POWER_SYS_GetErrorHandler` (`void`)
 This function returns the last failed notification callback configuration structure.
- `bool POWER_SYS_GetVeryLowPowerModeStatus` (`void`)
 This function returns whether very low power mode is running.

- bool [POWER_SYS_GetLowLeakageWakeupResetStatus](#) (void)
This function returns whether reset was caused by low leakage wake up.
- bool [POWER_SYS_GetAckIsolation](#) (void)
Gets the acknowledge isolation flag.
- void [POWER_SYS_ClearAckIsolation](#) (void)
Clears the acknowledge isolation flag.

45.2 Data Structure Documentation

45.2.1 struct power_manager_user_config_t

This structure defines Kinetis power mode with additional power options and specifies transition to and out of this mode. Application may define multiple power modes and switch between them. List of defined power modes is passed to the Power manager during initialization as an array of references to structures of this type (see [POWER_SYS_Init\(\)](#)). Power modes can be switched by calling [POWER_SYS_SetMode\(\)](#) which accepts index to the list of power modes passed during manager initialization. Currently used power mode can be retrieved by calling [POWER_SYS_GetLastMode\(\)](#), which returns index of the current power mode, or by [POWER_SYS_GetLastModeConfig\(\)](#), which returns reference to the structure of current mode. List of power mode configuration structure members depends on power options available for specific chip. Complete list contains: mode - Kinetis power mode. List of available modes is chip-specific. See `power_manager_modes_t` list of modes. This item is common for all Kinetis chips. `sleepOnExitOption` - Controls whether the sleep-on-exit option value is used (when set to true) or ignored (when set to false). See `sleepOnExitValue`. This item is common for all Kinetis chips. `sleepOnExitValue` - When set to true, ARM core returns to sleep (Kinetis wait modes) or deep sleep state (Kinetis stop modes) after interrupt service finishes. When set to false, core stays woken-up. This item is common for all Kinetis chips. `lowPowerWakeUpOnInterruptOption` - Controls whether the wake-up-on-interrupt option value is used (when set to true) or ignored (when set to false). See `lowPowerWakeUpOnInterruptValue`. This item is chip-specific. `lowPowerWakeUpOnInterruptValue` - When set to true, system exits to Run mode when any interrupt occurs while in Very low power run, Very low power wait or Very low power stop mode. This item is chip-specific. `powerOnResetDetectionOption` - Controls whether the power on reset detection option value is used (when set to true) or ignored (when set to false). See `powerOnResetDetectionValue`. This item is chip-specific. `powerOnResetDetectionValue` - When set to true, power on reset detection circuit is enabled in Very low leakage stop 0 mode. When set to false, circuit is disabled. This item is chip-specific. `RAM2PartitionOption` - Controls whether the RAM2 partition power option value is used (when set to true) or ignored (when set to false). See `RAM2PartitionValue`. This item is chip-specific. `RAM2PartitionValue` - When set to true, RAM2 partition content is retained through Very low leakage stop 2 mode. When set to false, RAM2 partition power is disabled and memory content lost. This item is chip-specific. `lowPowerOscillatorOption` - Controls whether the Low power oscillator power option value is used (when set to true) or ignored (when set to false). See `lowPowerOscillatorValue`. This item is chip-specific. `lowPowerOscillatorValue` - When set to true, the 1 kHz Low power oscillator is enabled in any Low leakage or Very low leakage stop mode. When set to false, oscillator is disabled in these modes. This item is chip-specific.

45.2.2 struct power_manager_notify_struct_t

Data Fields

- uint8_t targetPowerConfigIndex
Target power configuration index.
- power_manager_user_config_t * targetPowerConfigPtr
Pointer to target power configuration.
- power_manager_policy_t policy
Clock transition policy.
- power_manager_notify_t notifyType
Clock notification type.

45.2.2.0.0.50 Field Documentation

45.2.2.0.0.50.1 uint8_t power_manager_notify_struct_t::targetPowerConfigIndex

45.2.2.0.0.50.2 power_manager_policy_t power_manager_notify_struct_t::policy

45.2.2.0.0.50.3 power_manager_notify_t power_manager_notify_struct_t::notifyType

45.2.3 struct power_manager_callback_user_config_t

This structure holds configuration of callbacks passed to the Power manager during its initialization. Callbacks of this type are expected to be statically allocated. This structure contains 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

45.2.4 struct power_manager_state_t

Power manager internal structure. Contains data necessary for Power manager proper function. Stores references to registered power mode configurations, callbacks, information about their numbers and other internal data. This structure is statically allocated and initialized after `POWER_SYS_Init()` call.

Data Fields

- power_manager_user_config_t *(* configs)[]
Pointer to power configure table.
- uint8_t configsNumber
Number of power configurations.
- power_manager_callback_user_config_t *(* staticCallbacks)[]
Pointer to callback table.
- uint8_t staticCallbacksNumber
Max.
- uint8_t errorCallbackIndex
Index of callback returns error.

- `uint8_t currentConfig`
Index of current configuration.

45.2.4.0.0.51 Field Documentation

45.2.4.0.0.51.1 `power_manager_user_config_t*(* power_manager_state_t::configs)[]`

45.2.4.0.0.51.2 `power_manager_callback_user_config_t*(* power_manager_state_t::staticCallbacks)[]`

45.2.4.0.0.51.3 `uint8_t power_manager_state_t::staticCallbacksNumber`

number of callback configurations

45.2.4.0.0.51.4 `uint8_t power_manager_state_t::errorCallbackIndex`

45.2.4.0.0.51.5 `uint8_t power_manager_state_t::currentConfig`

45.3 Typedef Documentation

45.3.1 **typedef void power_manager_callback_data_t**

Reference to data of this type is passed during callback registration. The reference is part of the [power_manager_callback_user_config_t](#) structure and is passed to the callback during power mode change notifications.

45.3.2 **typedef power_manager_error_code_t(* power_manager_callback_t)(power_manager_notify_struct_t *notify, power_manager_callback_data_t *dataPtr)**

Declaration of callback. It is common for registered callbacks. Reference to function of this type is part of [power_manager_callback_user_config_t](#) callback configuration structure. Depending on callback type, function of this prototype is called during power mode change (see [POWER_SYS_SetMode\(\)](#)) before the mode change, after it or in both cases to notify about the change progress (see [power_manager_callback_type_t](#)). When called, type of the notification is passed as parameter along with reference to entered power mode configuration structure (see [power_manager_notify_struct_t](#)) and any data passed during the callback registration (see [power_manager_callback_data_t](#)). When notified before the mode change, depending on the power mode change policy (see [power_manager_policy_t](#)) the callback may deny the mode change by returning any error code different from kPowerManagerSuccess (see [POWER_SYS_SetMode\(\)](#)).

Enumeration Type Documentation

Parameters

<i>notify</i>	Notification structure.
<i>dataPtr</i>	Callback data. Refers to the data passed during callback registration. Intended to pass any driver or application data such as internal state information.

Returns

An error code or kPowerManagerSuccess.

45.4 Enumeration Type Documentation

45.4.1 enum power_manager_modes_t

Defines power mode. Used in the power mode configuration structure ([power_manager_user_config_t](#)). From ARM core perspective, Power modes can be generally divided to run modes (High speed run, Run and Very low power run), sleep (Wait and Very low power wait) and deep sleep modes (all Stop modes). List of power modes supported by specific chip along with requirements for entering and exiting of these modes can be found in chip documentation. List of all supported power modes:

- kPowerManagerHsrun - High speed run mode. Chip-specific.
- kPowerManagerRun - Run mode. All Kinetis chips.
- kPowerManagerVlpr - Very low power run mode. All Kinetis chips.
- kPowerManagerWait - Wait mode. All Kinetis chips.
- kPowerManagerVlpw - Very low power wait mode. All Kinetis chips.
- kPowerManagerStop - Stop mode. All Kinetis chips.
- kPowerManagerVlps - Very low power stop mode. All Kinetis chips.
- kPowerManagerPstop1 - Partial stop 1 mode. Chip-specific.
- kPowerManagerPstop2 - Partial stop 2 mode. Chip-specific.
- kPowerManagerLls - Low leakage stop mode. All Kinetis chips.
- kPowerManagerLls2 - Low leakage stop 2 mode. Chip-specific.
- kPowerManagerLls3 - Low leakage stop 3 mode. Chip-specific.
- kPowerManagerVlls0 - Very low leakage stop 0 mode. Chip-specific.
- kPowerManagerVlls1 - Very low leakage stop 1 mode. All Kinetis chips.
- kPowerManagerVlls2 - Very low leakage stop 2 mode. All Kinetis chips.
- kPowerManagerVlls3 - Very low leakage stop 3 mode. All Kinetis chips.

Enumerator

kPowerManagerRun Run mode. All Kinetis chips.

kPowerManagerVlpr Very low power run mode. All Kinetis chips.

kPowerManagerWait Wait mode. All Kinetis chips.

kPowerManagerVlpw Very low power wait mode. All Kinetis chips.

kPowerManagerStop Stop mode. All Kinetis chips.

kPowerManagerVlps Very low power stop mode. All Kinetis chips.

kPowerManagerVlls1 Very low leakage stop 1 mode. All Kinetis chips.

kPowerManagerVlls3 Very low leakage stop 3 mode. All Kinetis chips.

45.4.2 enum power_manager_error_code_t

Used as return value of Power manager functions.

Enumerator

kPowerManagerSuccess Success.

kPowerManagerError Some error occurs.

kPowerManagerErrorOutOfRange Configuration index out of range.

kPowerManagerErrorSwitch Error occurs during mode switch.

kPowerManagerErrorNotificationBefore Error occurs during send "BEFORE" notification.

kPowerManagerErrorNotificationAfter Error occurs during send "AFTER" notification.

kPowerManagerErrorClock Error occurs due to wrong clock setup for power modes.

45.4.3 enum power_manager_policy_t

Define whether the power mode change is forced or not. Used to specify whether the mode switch initiated by the [POWER_SYS_SetMode\(\)](#) depends on the callback notification results. For **kPowerManagerPolicyForcible** the power mode is changed regardless of the results, while **kPowerManagerPolicyAgreement** policy is used the [POWER_SYS_SetMode\(\)](#) is exited when any of the callbacks returns error code. See also [POWER_SYS_SetMode\(\)](#) description.

Enumerator

kPowerManagerPolicyAgreement [POWER_SYS_SetMode\(\)](#) method is exited when any of the callbacks returns error code.

kPowerManagerPolicyForcible Power mode is changed regardless of the results.

45.4.4 enum power_manager_notify_t

Used to notify registered callbacks

Enumerator

kPowerManagerNotifyRecover Notify IP to recover to previous work state.

kPowerManagerNotifyBefore Notify IP that system will change power setting.

kPowerManagerNotifyAfter Notify IP that have changed to new power setting.

Function Documentation

45.4.5 enum power_manager_callback_type_t

Used in the callback configuration structures ([power_manager_callback_user_config_t](#)) to specify when the registered callback will be called during power mode change initiated by [POWER_SYS_SetMode\(\)](#). Callback can be invoked in following situations:

- before the power mode change (Callback return value can affect [POWER_SYS_SetMode\(\)](#) execution. Refer to the [POWER_SYS_SetMode\(\)](#) and [power_manager_policy_t](#) documentation).
- after entering one of the run modes or after exiting from one of the (deep) sleep power modes back to the run mode.
- after unsuccessful attempt to switch power mode

Enumerator

kPowerManagerCallbackBefore Before callback.

kPowerManagerCallbackAfter After callback.

kPowerManagerCallbackBeforeAfter Before-After callback.

45.5 Function Documentation

45.5.1 power_manager_error_code_t POWER_SYS_Init ([power_manager_user_config_t](#) const [\(*\) powerConfigsPtr\[\]](#), [uint8_t](#) [configsNumber](#), [power_manager_callback_user_config_t](#) const [\(*\) callbacksPtr\[\]](#), [uint8_t](#) [callbacksNumber](#))

This function initializes the Power manager and its run-time state structure. Reference to an array of Power mode configuration structures has to be passed as a parameter along with a parameter specifying its size. At least one power mode configuration is required. Optionally, reference to the array of predefined callbacks can be passed with its size parameter. For details about callbacks, refer to the [power_manager_callback_user_config_t](#). As Power manager stores only references to array of these structures, they have to exist while Power manager is used. It is expected that prior to the [POWER_SYS_Init\(\)](#) call the write-once protection register was configured appropriately allowing for entry to all required low power modes. The following is an example of how to set up two power modes and three callbacks, and initialize the Power manager with structures containing their settings. The example shows two possible ways the configuration structures can be stored (ROM or RAM), although it is expected that they will be placed in the read-only memory to save the RAM space. (Note: In the example it is assumed that the programmed chip doesn't support any optional power options described in the [power_manager_user_config_t](#)) :

```
const power\_manager\_user\_config\_t waitConfig = {
    kPowerManagerVlpw,
    true,
    true,
};

const power\_manager\_callback\_user\_config\_t callbackCfg0 = {
    callback0,
    kPowerManagerCallbackBefore,
    &callback_data0
};
```

```

const power_manager_callback_user_config_t callbackCfg1 = {
    callback1,
    kPowerManagerCallbackAfter,
    &callback_data1
};

const power_manager_callback_user_config_t callbackCfg2 = {
    callback2,
    kPowerManagerCallbackBeforeAfter,
    &callback_data2
};

const power_manager_callback_user_config_t * const callbacks[] = {&
    callbackCfg0, &callbackCfg1, &callbackCfg2};

void main(void)
{
    power_manager_user_config_t idleConfig;
    power_manager_user_config_t *powerConfigs[] = {&idleConfig, &waitConfig};

    idleConfig.mode = kPowerManagerVlps;
    idleConfig.sleepOnExitOption = true;
    idleConfig.sleepOnExitValue = false;

    POWER_SYS_Init(&powerConfigs, 2U, &callbacks, 3U);

    POWER_SYS_SetMode(0U, kPowerManagerPolicyAgreement);
}
*

```

Parameters

<i>powerConfigs-Ptr</i>	A pointer to an array with references to all power configurations which will be handled by Power manager.
<i>configsNumber</i>	Number of power configurations. Size of powerConfigsPtr array.
<i>callbacksPtr</i>	A pointer to an array with references to callback configurations. If there are no callbacks to register during Power manager initialization, use NULL value.
<i>callbacks-Number</i>	Number of registered callbacks. Size of callbacksPtr array.

Returns

An error code or kPowerManagerSuccess.

45.5.2 power_manager_error_code_t POWER_SYS_Deinit(void)

Returns

An error code or kPowerManagerSuccess.

Function Documentation

45.5.3 power_manager_error_code_t POWER_SYS_SetMode (uint8_t *powerModelIndex*, power_manager_policy_t *policy*)

This function switches to one of the defined power modes. Requested mode number is passed as an input parameter. This function notifies all registered callback functions before the mode change (using kPowerManagerCallbackBefore set as callback type parameter), sets specific power options defined in the power mode configuration and enters the specified mode. In case of run modes (for example, Run, Very low power run, or High speed run), this function also invokes all registered callbacks after the mode change (using kPowerManagerCallbackAfter). In case of sleep or deep sleep modes, if the requested mode is not exited through a reset, these notifications are sent after the core wakes up. Callbacks are invoked in the following order: All registered callbacks are notified ordered by index in the callbacks array (see callbacksPtr parameter of [POWER_SYS_Init\(\)](#)). The same order is used for before and after switch notifications. The notifications before the power mode switch can be used to obtain confirmation about the change from registered callbacks. If any registered callback denies the power mode change, further execution of this function depends on mode change policy: the mode change is either forced (kPowerManagerPolicyForcible) or exited (kPowerManagerPolicyAgreement). When mode change is forced, the result of the before switch notifications are ignored. If agreement is required, if any callback returns an error code then further notifications before switch notifications are cancelled and all already notified callbacks are re-invoked with kPowerManagerCallbackAfter set as callback type parameter. 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 [POWER_SYS_GetErrorCallback\(\)](#) can be used to get it. Regardless of the policies, if any callback returned an error code, an error code denoting in which phase the error occurred is returned when [POWER_SYS_SetMode\(\)](#) exits. It is possible to enter any mode supported by the processor. Refer to the chip reference manual for list of available power modes. If it is necessary to switch into intermediate power mode prior to entering requested mode (for example, when switching from Run into Very low power wait through Very low power run mode), then the intermediate mode is entered without invoking the callback mechanism.

Parameters

<i>powerModelIndex</i>	Requested power mode represented as an index into array of user-defined power mode configurations passed to the POWER_SYS_Init() .
<i>policy</i>	Transaction policy

Returns

An error code or kPowerManagerSuccess.

45.5.4 power_manager_error_code_t POWER_SYS_GetLastMode (uint8_t * *powerModelIndexPtr*)

This function returns index of power mode which was set using [POWER_SYS_SetMode\(\)](#) as the last one. If the power mode was entered even though some of the registered callback denied the mode change, or

if any of the callbacks invoked after the entering/restoring run mode failed, then the return code of this function has kPowerManagerError value.

Parameters

<i>powerMode-IndexPtr</i>	Power mode which has been set represented as an index into array of power mode configurations passed to the POWER_SYS_Init() .
---------------------------	--

Returns

An error code or kPowerManagerSuccess.

45.5.5 **power_manager_error_code_t POWER_SYS_GetLastModeConfig (power_manager_user_config_t ** *powerModePtr*)**

This function returns reference to configuration structure which was set using [POWER_SYS_SetMode\(\)](#) as the last one. If the current power mode was entered even though some of the registered callback denied the mode change, or if any of the callbacks invoked after the entering/restoring run mode failed, then the return code of this function has kPowerManagerError value.

Parameters

<i>powerModePtr</i>	Pointer to power mode configuration structure of power mode set as last one.
---------------------	--

Returns

An error code or kPowerManagerSuccess.

45.5.6 **power_manager_modes_t POWER_SYS_GetCurrentMode (void)**

This function reads hardware settings and returns currently running power mode. Generally, this function can return only kPowerManagerRun, kPowerManagerVlpr or kPowerManagerHsrun value.

Returns

Currently used run power mode.

45.5.7 **uint8_t POWER_SYS_GetErrorCallbackIndex (void)**

This function returns index of the last callback that failed during the power mode switch while the last [POWER_SYS_SetMode\(\)](#) was called. If the last [POWER_SYS_SetMode\(\)](#) call ended successfully value equal to callbacks number is returned. Returned value represents index in the array of static call-backs.

Function Documentation

Returns

Callback index of last failed callback or value equal to callbacks count.

45.5.8 power_manager_callback_user_config_t* POWER_SYS_GetErrorCallback (void)

This function returns pointer to configuration structure of the last callback that failed during the power mode switch while the last [POWER_SYS_SetMode\(\)](#) was called. If the last [POWER_SYS_SetMode\(\)](#) call ended successfully value NULL is returned.

Returns

Pointer to the callback configuration which returns error.

45.5.9 bool POWER_SYS_GetVeryLowPowerModeStatus (void)

This function is used to detect whether very low power mode is running.

Returns

Returns true if processor runs in very low power mode, otherwise false.

45.5.10 bool POWER_SYS_GetLowLeakageWakeupResetStatus (void)

This function is used to check that processor exited low leakage power mode through reset.

Returns

Returns true if processor was reset by low leakage wake up, otherwise false.

45.5.11 bool POWER_SYS_GetAckIsolation (void)

This function is used to check certain peripherals and the I/O pads are in a latched state as a result of having been in a VLLS mode.

After recovery from VLLS, the LLWU continues to detect wake-up events until the user has acknowledged the wake-up via [POWER_SYS_ClearAckIsolation\(\)](#)

Returns

Returns true if ACK isolation is set.

45.5.12 void POWER_SYS_ClearAckIsolation(void)

This function clears the ACK Isolation flag. Clearing releases the I/O pads and certain peripherals to their normal run mode state.

After recovery from VLLS, the LLWU continues to detect wake-up events until the user has acknowledged the wake-up via [POWER_SYS_ClearAckIsolation\(\)](#)

Power Manager driver

45.6 Power Manager driver

Low Power Manager provides API to handle the device power modes. It also supports run-time switching between multiple power modes. Each power mode is described by configuration structures with multiple power-related options. Low Power Manager provides a notification mechanism for registered callbacks and API for static and dynamic callback registration.

45.6.1 Power Manager Overview

The Power Manager driver is developed on top of the SMC HAL, PMC HAL and RCM HAL.

This is an example to initialize the Power Manager:

~~~~~{.c}

```

/* Power modes configurations */
power_manager_user_config_t vlprConfig;
power_manager_user_config_t stopConfig;

power_manager_user_config_t *powerConfigs[] = {&vlprConfig, &stopConfig};

/* Definition of transition to and out these power modes */
vlprConfig.mode = kPowerManagerVlpr;
vlprConfig.policy = kPowerManagerPolicyAgreement;
vlprConfig.sleepOnExitValue = false;
vlprConfig.sleepOnExitOption = false;

stopConfig = vlprConfig;
stopConfig.mode = kPowerManagerStop;

/* Calling of init method */
POWER_SYS_Init(&powerConfigs, 2U, &callbacks, 1U);
}

```

~~~~~{.c}

This is an example to switch into a desired power mode:

~~~~~{.c}

```

#include "fsl_power_manager.h"

/* Index into array containing configuration of very low power run mode - vlpr */
#define MODE_VLPR 0U

/* Initialization */
power_manager_user_config_t vlprConfig;
...
...
power_manager_user_config_t *powerConfigs[1] = {&vlprConfig};

/* Switch to required power mode */
power_manager_error_code_t ret = POWER_SYS_SetMode(MODE_VLPR);

if (ret != kPowerManagerSuccess)
{
    printf("POWER_SYS_SetMode(powerMode5) returns : %u\n\r", ret);
}

```

~~~~~{.c}


Chapter 46

Utilities for the Kinetis SDK

46.1 Overview

Modules

- [Debug_console](#)

This part describes the programming interface of the debug console driver.

46.2 Debug_console

This chapter describes the programming interface of the debug console driver.

Chapter 47

OS Abstraction Layer (OSA)

The Kinetis SDK provides the timer services for all OSA layers.

47.1 Overview

Modules

- Bare Metal Abstraction Layer

The Kinetis SDK provides the Bare Metal Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

- FreeRTOS Abstraction Layer

The Kinetis SDK provides the FreeRTOS Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

- MQX Abstraction Layer

The Kinetis SDK provides the MQX Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

- μC/OS-II Abstraction Layer

The Kinetis SDK provides the μC/OS-II Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

- μC/OS-III Abstraction Layer

The Kinetis SDK provides the μC/OS-III Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

Enumerations

- enum `osa_status_t` {
 `kStatus_OSA_Success` = 0U,
 `kStatus_OSA_Error` = 1U,
 `kStatus_OSA_Timeout` = 2U,
 `kStatus_OSA_Idle` = 3U }

Defines the return status of OSA's functions.

- enum `osa_event_clear_mode_t` {
 `kEventAutoClear` = 0U,
 `kEventManualClear` = 1U }

The event flags are cleared automatically or manually.

- enum `osa_critical_part_mode_t` {
 `kCriticalLockSched` = 0U,
 `kCriticalDisableInt` = 1U }

Locks the task scheduler or disables interrupt in critical part.

Counting Semaphore

- `osa_status_t OSA_SemaCreate (semaphore_t *pSem, uint8_t initialValue)`

Overview

- **osa_status_t OSA_SemaWait (semaphore_t *pSem, uint32_t timeout)**
Pending a semaphore with timeout.
- **osa_status_t OSA_SemaPost (semaphore_t *pSem)**
Signals for someone waiting on the semaphore to wake up.
- **osa_status_t OSA_SemaDestroy (semaphore_t *pSem)**
Destroys a previously created semaphore.

Mutex

- **osa_status_t OSA_MutexCreate (mutex_t *pMutex)**
Create an unlocked mutex.
- **osa_status_t OSA_MutexLock (mutex_t *pMutex, uint32_t timeout)**
Waits for a mutex and locks it.
- **osa_status_t OSA_MutexUnlock (mutex_t *pMutex)**
Unlocks a previously locked mutex.
- **osa_status_t OSA_MutexDestroy (mutex_t *pMutex)**
Destroys a previously created mutex.

Event signalling

- **osa_status_t OSA_EventCreate (event_t *pEvent, osa_event_clear_mode_t clearMode)**
Initializes an event object with all flags cleared.
- **osa_status_t OSA_EventWait (event_t *pEvent, event_flags_t flagsToWait, bool waitAll, uint32_t timeout, event_flags_t *setFlags)**
Waits for specified event flags to be set.
- **osa_status_t OSA_EventSet (event_t *pEvent, event_flags_t flagsToSet)**
Sets one or more event flags.
- **osa_status_t OSA_EventClear (event_t *pEvent, event_flags_t flagsToClear)**
Clears one or more flags.
- **event_flags_t OSA_EventGetFlags (event_t *pEvent)**
Gets event flags status.
- **osa_status_t OSA_EventDestroy (event_t *pEvent)**
Destroys a previously created event object.

Task management

- **osa_status_t OSA_TaskCreate (task_t task, uint8_t *name, uint16_t stackSize, task_stack_t *stackMem, uint16_t priority, task_param_t param, bool usesFloat, task_handler_t *handler)**
Creates a task.
- **osa_status_t OSA_TaskDestroy (task_handler_t handler)**
Destroys a previously created task.
- **osa_status_t OSA_TaskYield (void)**
Puts the active task to the end of scheduler's queue.
- **task_handler_t OSA_TaskGetHandler (void)**
Gets the handler of active task.
- **uint16_t OSA_TaskGetPriority (task_handler_t handler)**
Gets the priority of a task.
- **osa_status_t OSA_TaskSetPriority (task_handler_t handler, uint16_t priority)**
Sets the priority of a task.

Message queues

- `msg_queue_handler_t OSA_MsgQCreate (msg_queue_t *queue, uint16_t message_number, uint16_t message_size)`
Initializes a message queue.
- `osa_status_t OSA_MsgQPut (msg_queue_handler_t handler, void *pMessage)`
Puts a message at the end of the queue.
- `osa_status_t OSA_MsgQGet (msg_queue_handler_t handler, void *pMessage, uint32_t timeout)`
Reads and remove a message at the head of the queue.
- `osa_status_t OSA_MsgQDestroy (msg_queue_handler_t handler)`
Destroys a previously created queue.

Memory Management

- `void * OSA_MemAlloc (size_t size)`
Reserves the requested amount of memory in bytes.
- `void * OSA_MemAllocZero (size_t size)`
Reserves the requested amount of memory in bytes and initializes it to 0.
- `osa_status_t OSA_MemFree (void *ptr)`
Releases the memory previously reserved.

Time management

- `void OSA_TimeDelay (uint32_t delay)`
Delays execution for a number of milliseconds.
- `uint32_t OSA_TimeGetMsec (void)`
Gets the current time since system boot in milliseconds.

Interrupt management

- `osa_status_t OSA_InstallIntHandler (int32_t IRQNumber, void(*handler)(void))`
Installs the interrupt handler.

Critical section

- `void OSA_EnterCritical (osa_critical_part_mode_t mode)`
Enters the critical part to ensure some code is not preempted.
- `OSA_ExitCritical (osa_critical_part_mode_t mode)`
Exits the critical part.

OSA initialize

- `osa_status_t OSA_Init (void)`
Initializes the RTOS services.
- `osa_status_t OSA_Start (void)`
Starts the RTOS.

47.1.1 OS Abstraction Layer

Overview

Overview

Operating System Abstraction layer (OSA) provides a common set of services for drivers and applications so that they can work with or without the operating system. OSA provides services that abstract most of the OS kernel functionality. These services can either be mapped to the target OS functions directly, or implemented by OSA when no OS is used (bare metal) or when the service does not exist in the target OS. Freescale Kinetis SDK implements the OS abstraction layer for MQX™ RTOS, Free RTOS, μC/OS, and for no OS usage (bare metal). The bare metal OS abstraction implementation is selected as the default option.

OSA provides these services: task management, semaphore, mutex, event, message queue, memory allocator, critical part, and time functions.

Task Management

With OSA, applications can create and destroy tasks dynamically. These services are mapped to the task functions of RTOSes. For bare metal, a function poll mechanism simulates a task scheduler.

OSA supports task priorities 0~15, where priority 0 is the highest priority and priority 15 is the lowest priority.

To create a task, applications must prepare different resources on different RTOSes. For example, μC-/OS-II and μC/OS-III need pre-allocated task stack while other RTOSes do not need this. The μC/OS-III needs pre-allocated task control block OS_TCB while other RTOSes do not. To mask the differences, OSA uses a macro OSA_TASK_DEFINE to prepare resources for task creation. Then the function [OSA_TaskCreate\(\)](#) creates a task based on the resources. This method makes it easy to use a copy of code on different creates a task based on the resources. This method makes it easy to use a copy of code on different RTOSes. However, it is not mandatory to use the OSA_TASK_DEFINE. Applications can also prepare the resources manually. There are two methods to create a task:

1. Use the OSA_TASK_DEFINE macro and the function [OSA_TaskCreate\(\)](#). The macro OSA_TASK_DEFINE declares a task handler and task stack statically. The function [OSA_TaskCreate\(\)](#) creates task base-on the resources declared by OSA_TASK_DEFINE.

This is an example code to create a task using method 1:

```
// Define the task with entry function task_func.  
OSA_TASK_DEFINE(task_func, TASK_STACK_SIZE);  
  
void main(void)  
{  
    task_param_t parameter;  
  
    // Create the task.  
    OSA_TaskCreate(task_func, // Task function.  
                  "my_task", // Task name.  
                  TASK_STACK_SIZE, // Stack size.  
                  task_func_stack, // Stack address.  
                  TASK_PRIO, // Task priority.  
                  parameter, // Parameter.  
                  false, // Use float register or not.  
                  &task_func_task_handler); // Task handler.  
    // ...  
}
```

2. Prepare resources manually, then use the function [OSA_TaskCreate\(\)](#) to create a task.

For example:

```

task_param_t parameter;
task_handler_t task_handler;

#if defined(FSL_RTOS_UCOSII)
    task_stack_t task_stack[TASK_STACK_SIZE/sizeof(task_stack_t)];
    OSA_TaskCreate(task_func,                                // Task function.
                   "my_task",                           // Task name.
                   TASK_STACK_SIZE,                    // Stack size.
                   task_stack,                         // Stack address.
                   TASK_PRIO,                          // Task priority.
                   parameter,                         // Parameter.
                   false,                             // Use float register or not.
                   &task_handler);                     // Task handler.

#elif defined(FSL_RTOS_UCOSIII)
    task_stack_t task_stack[TASK_STACK_SIZE/sizeof(task_stack_t)];
    OS_TCB TCB_task;
    task_handler = &TCB_task;
    OSA_TaskCreate(task_func,                                // Task function.
                   "my_task",                           // Task name.
                   TASK_STACK_SIZE,                    // Stack size.
                   task_stack,                         // Stack address.
                   TASK_PRIO,                          // Task priority.
                   parameter,                         // Parameter.
                   false,                             // Use float register or not.
                   &task_handler);                     // Task handler.

#else // For MQX, FreeRTOS and bare metal.
    OSA_TaskCreate(task_func,                                // Task function.
                   "my_task",                           // Task name.
                   TASK_STACK_SIZE,                    // Stack size.
                   NULL,                             // Stack address.
                   TASK_PRIO,                          // Task priority.
                   parameter,                         // Parameter.
                   false,                             // Use float register or not.
                   &task_handler);                     // Task handler.
#endif

```

Method 1 is easy to use. The disadvantage is that one task function can only create one task instance. Method 2 can create multiple task instances using one task function, but the code must be divided by macros for different RTOSes. Applications can choose either method according to requirements.

After a task is created successfully, task handler can be used to manage the task, for example, get or set task priority, destroy task and so on.

If task is not used any more, use the [OSA_TaskDestroy\(\)](#) function to destroy the task. If the task function does not contain an infinite loop, or in other words, the task function returns, call the [OSA_TaskDestroy\(\)](#) function at the end of the task function.

Semaphore

The OSA provides the drivers and applications with a counting semaphore. It can be used either to synchronize tasks or to synchronize a task and an ISR.

A semaphore must be initialized with the [OSA_SemaCreate\(\)](#) function before using. The semaphore can be initialized with an initial value. When the semaphore is not used any more, use the [OSA_SemaDestroy\(\)](#) function to destroy it.

Note that only one task should wait per semaphore. If multiple tasks are waiting per semaphore, different RTOSes may have different behaviors.

Overview

This is an example code to create and destroy a semaphore:

```
semaphore_t sem;

// Initialize the semaphore with initial value.
OSA_SemaCreate(&sem, 0);

// Destroy the semaphore.
OSA_SemaDestroy(&sem);
```

The function [OSA_SemaWait\(\)](#) waits a semaphore within the timeout (in milliseconds). Passing a value 0 as a timeout means return immediately and passing the OSA_WAIT_FOREVER means wait indefinitely. This function should not be used in the ISR.

The function [OSA_SemaPost\(\)](#) wakes up task which is waiting for the semaphore.

Mutex

A mutex is used for the mutual exclusion of tasks when they access a shared resource. OSA provides a non-recursive mutex, which means a task cannot try to lock a mutex it has already locked.

A mutex must be initialized to an unlocked status with the [OSA_MutexCreate\(\)](#) function before using. When the mutex is not used any more, use the [OSA_MutexDestroy\(\)](#) function to destroy it.

This is example code to create and destroy a mutex:

```
mutex_t mutex;

// Initialize the mutex
OSA_MutexCreate(&mutex);

// Destroy the mutex
OSA_MutexDestroy(&mutex);
```

The function [OSA_MutexLock\(\)](#) waits to lock a mutex within the timeout (in milliseconds). Passing a value 0 as a timeout means return immediately and passing the OSA_WAIT_FOREVER means waiting indefinitely.

The function [OSA_MutexUnlock\(\)](#) unlocks a mutex which is locked by the current task.

Event

Event is used for a one-consumer-multi-producer model. It means that there should be only one task waiting for the event. If multiple tasks are waiting on one event, different RTOSes may have different behaviors.

OSA provides two types of events:

1. Auto-clear, which occurs when some task has get flags it is waiting for. These flags are cleared automatically.
2. Manual-clear, which means that the flags could only be cleared manually.

The clear mode is a property of an event. Once an event is created, the clear mode can't be changed.

An event must be initialized with the [OSA_EventCreate\(\)](#) function before using. When it is created, its flags are all cleared. When the event is not used any more, use the [event_destory\(\)](#) function to destroy it.

This is example code to create and destroy an event object:

```
event_t event;

// Initialize the event
OSA_EventCreate(&event, kEventAutoClear);

// Destroy the event
OSA_EventDestroy(&event);
```

The function [OSA_EventWait\(\)](#) waits for specified flags of an event with the timeout (in milliseconds). Passing a value 0 as a timeout means return immediately and passing the [OSA_WAIT_FOREVER](#) means wait indefinitely. This function can be configured to wait for all specified flags or wait for any one flag in specified flags. The parameter setFlags saves the flags which wake up the waiting task. This function should not be used in the ISR.

The functions [OSA_EventSet\(\)](#) and [OSA_EventClear\(\)](#) are used to set and clear specified flags of an event. The function [OSA_EventGetFlags\(\)](#) is used to get current event flags.

Message Queue

OSA provides the FIFO message queue. All messages in a queue have the same size. Message queue holds an internal memory area to save messages. While putting the message, the message entity is copied to this internal memory area. While getting message, message is copied from the internal memory area.

Please note that there should be only one task waiting on one message queue, if multiple tasks are waiting on one message queue, different RTOSes may have different behaviors.

To create a message queue, use [MSG_QUEUE_DECLARE\(\)](#) and [OSA_MsgQCreate\(\)](#) functions as shown here:

```
// Declare the message queue.
MSG_QUEUE_DECLARE(my_message, msg_num, msg_size);

void main(void)
{
    msg_queue_handler_t handler;
    handler = OSA_MsgQCreate(my_message, msg_num, msg_size);
    // ...
}
```

Note that the parameter `message_size` is in words and not in bytes. It means that the message queue can only transfer messages of multiple-byte size.

The function [OSA_MsgQPut\(\)](#) puts a message to the queue. If the queue is full, an error returns.

The function [OSA_MsgQGet\(\)](#) waits for a timeout in milliseconds to get the message from the queue. Passing a value 0 as a timeout means return immediately and passing the [OSA_WAIT_FOREVER](#) means wait indefinitely. This function should not be used in the ISR.

If the queue is not used any more, use the [OSA_MsgQDestroy\(\)](#) function to destroy it.

Overview

Critical Section

OSA provides two types of critical sections. The first type disables the interrupt while the second type only disables the scheduler to stop the task preemption.

Memory Allocator

The function [OSA_MemAlloc\(\)](#) allocates memory with specified size. The function [OSA_MemAllocZero\(\)](#) allocates and cleans the memory. The function [OSA_MemFree\(\)](#) frees the memory. For RTOSes that have internal memory manager, such as the MQX, OSA maps these functions directly. For other RTOSes or bare metal, the standard functions malloc/calloc/free are used.

Time Functions

OSA only provides two time functions, [OSA_TimeDelay\(\)](#) and [OSA_TimeGetMsec\(\)](#). The function [OSA_TimeDelay\(\)](#) delays specified time in milliseconds, while the function [OSA_TimeGetMsec\(\)](#) gets the system time in milliseconds since POR.

Interrupt priority

For some RTOSes, a proper interrupt priority must be set if system services are called in this interrupt service routine.

For MQX RTOS, follow these criteria:

1. Interrupt priority must be an even number.
2. Interrupt priority $\geq 2 * \text{MQX_HARDWARE_INTERRUPT_LEVEL_MAX}$.

For FreeRTOS, the interrupt priority is defined in the configuration file FreeRTOSConfig.h. In the current configuration, priority 1~15 can be used for the ARM Cortex®-M4. See the FreeRTOS' official documents for details.

OSA initialization

To initialize and start RTOSes, OSA uses abstract functions [OSA_Init\(\)](#) and [OSA_Start\(\)](#).

This example shows how to use [OSA_Init\(\)](#) and [OSA_Start\(\)](#) functions:

```
#include <fsl_os_abstraction.h>

void task_func(task_param_t param)
{
    //...
}

OSA_TASK_DEFINE(task_func, 512);

#if defined(FSL_RTOS_MQX)
void Main_Task(uint32_t param);
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
    { 1L, Main_Task, 256L, MQX_MAIN_TASK_PRIORITY, "Main", MQX_AUTO_START_TASK},
    { 0L, 0L, 0L, 0L, 0L, 0L }
};
#endif
```

```

#if defined(FSL_RTOS_MQX)
void Main_Task(uint32_t param)
#else
void main(void)
#endif
{
    OSA_Init();
    // Other application initialize functions.
    // ...
    OSA_TaskCreate(task_func,           // Task function.
                  "my_task",        // Task name.
                  512,              // Stack size.
                  task_func_stack, // Stack address.
                  5,                // Task priority.
                  (task_param_t)0, // Parameter.
                  false,             // Use float register or not.
                  &task_func_task_handler); // Task handler.
    OSA_Start();
}

```

Note that, for other RTOSes, the task scheduler is started up by the `OSA_Start()` function, but for the MQX RTOS, the task scheduler was started up before the `OSA_Init()` function call. When creating tasks in the `Main_Task()` function with the MQX RTOS, the newly created task runs immediately if it has the highest priority. However, for other RTOSes, all newly created tasks do not run until the `OSA_Start()` function executes.

47.2 Enumeration Type Documentation

47.2.1 enum osa_status_t

Enumerator

- kStatus_OSA_Success*** Success.
- kStatus_OSA_Error*** Failed.
- kStatus_OSA_Timeout*** Timeout occurs while waiting.
- kStatus_OSA_Idle*** Used for bare metal only, the wait object is not ready and timeout still not occur.

47.2.2 enum osa_event_clear_mode_t

Enumerator

- kEventAutoClear*** The flags of the event will be cleared automatically.
- kEventManualClear*** The flags of the event will be cleared manually.

47.2.3 enum osa_critical_section_mode_t

Enumerator

- kCriticalSectionSched*** Lock scheduler in critical part.
- kCriticalSectionDisableInt*** Disable interrupt in critical selection.

Function Documentation

47.3 Function Documentation

47.3.1 `osa_status_t OSA_SemaCreate (semaphore_t * pSem, uint8_t initialValue)`

This function creates a semaphore and sets the value to the parameter initialValue.

Parameters

| | |
|------------------|---|
| <i>pSem</i> | Pointer to the semaphore. |
| <i>initValue</i> | Initial value the semaphore will be set to. |

Return values

| | |
|----------------------------|--|
| <i>kStatus_OSA_Success</i> | The semaphore is created successfully. |
| <i>kStatus_OSA_Error</i> | The semaphore can not be created. |

Example:

```
semaphore_t mySem;
OSA_SemaCreate(&mySem, 0);
```

47.3.2 osa_status_t OSA_SemaWait (semaphore_t * *pSem*, uint32_t *timeout*)

This function checks the semaphore's counting value. If it is positive, decreases it and returns kStatus_O- SA_Success. Otherwise, a timeout is used to wait.

Parameters

| | |
|----------------|--|
| <i>pSem</i> | Pointer to the semaphore. |
| <i>timeout</i> | The maximum number of milliseconds to wait if semaphore is not positive. Pass OS- A_WAIT_FOREVER to wait indefinitely, pass 0 will return kStatus_OSA_Timeout immediately. |

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | The semaphore is received. |
| <i>kStatus_OSA_Timeout</i> | The semaphore is not received within the specified 'timeout'. |
| <i>kStatus_OSA_Error</i> | An incorrect parameter was passed. |
| <i>kStatus_OSA_Idle</i> | The semaphore is not available and 'timeout' is not exhausted, This is only for bare metal. |

Note

With bare metal, a semaphore can not be waited by more than one task at the same time.

Example:

```
osa_status_t status;
status = OSA_SemaWait(&mySem, 100);
```

Function Documentation

```
switch(status)
{
    //...
}
```

47.3.3 osa_status_t OSA_SemaPost (semaphore_t * pSem)

Wakes up one task that is waiting on the semaphore. If no task is waiting, increases the semaphore's counting value.

Parameters

| | |
|-------------|-------------------------------------|
| <i>pSem</i> | Pointer to the semaphore to signal. |
|-------------|-------------------------------------|

Return values

| | |
|----------------------------|--|
| <i>kStatus_OSA_Success</i> | The semaphore is successfully signaled. |
| <i>kStatus_OSA_Error</i> | The object can not be signaled or invalid parameter. |

Example:

```
osa_status_t status;
status = OSA_SemaPost (&mySem);
switch(status)
{
    //...
}
```

47.3.4 osa_status_t OSA_SemaDestroy (semaphore_t * pSem)

Parameters

| | |
|-------------|--------------------------------------|
| <i>pSem</i> | Pointer to the semaphore to destroy. |
|-------------|--------------------------------------|

Return values

| | |
|----------------------------|--|
| <i>kStatus_OSA_Success</i> | The semaphore is successfully destroyed. |
| <i>kStatus_OSA_Error</i> | The semaphore can not be destroyed. |

Example:

```
osa_status_t status;
status = OSA_SemaDestroy (&mySem);
switch(status)
{
    //...
}
```

47.3.5 osa_status_t OSA_MutexCreate (mutex_t * *pMutex*)

This function creates a non-recursive mutex and sets it to unlocked status.

Function Documentation

Parameters

| | |
|---------------|-----------------------|
| <i>pMutex</i> | Pointer to the Mutex. |
|---------------|-----------------------|

Return values

| | |
|----------------------------|------------------------------------|
| <i>kStatus_OSA_Success</i> | The mutex is created successfully. |
| <i>kStatus_OSA_Error</i> | The mutex can not be created. |

Example:

```
mutex_t myMutex;
osa_status_t status;
status = OSA_MutexCreate(&myMutex);
switch (status)
{
    //...
}
```

47.3.6 **osa_status_t OSA_MutexLock (mutex_t * *pMutex*, uint32_t *timeout*)**

This function checks the mutex's status. If it is unlocked, locks it and returns the kStatus_OSA_Success. Otherwise, waits for a timeout in milliseconds to lock.

Parameters

| | |
|----------------|--|
| <i>pMutex</i> | Pointer to the Mutex. |
| <i>timeout</i> | The maximum number of milliseconds to wait for the mutex. If the mutex is locked, Pass the value OSA_WAIT_FOREVER will wait indefinitely, pass 0 will return k-Status_OSA_Timeout immediately. |

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | The mutex is locked successfully. |
| <i>kStatus_OSA_Timeout</i> | Timeout occurred. |
| <i>kStatus_OSA_Error</i> | Incorrect parameter was passed. |
| <i>kStatus_OSA_Idle</i> | The mutex is not available and 'timeout' is not exhausted, This is only for bare metal. |

Note

This is non-recursive mutex, a task can not try to lock the mutex it has locked.

Example:

```
osa_status_t status;
status = OSA_MutexLock(&myMutex, 100);
switch (status)
{
    //...
}
```

47.3.7 osa_status_t OSA_MutexUnlock (mutex_t * pMutex)

Parameters

| | |
|---------------|-----------------------|
| <i>pMutex</i> | Pointer to the Mutex. |
|---------------|-----------------------|

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | The mutex is successfully unlocked. |
| <i>kStatus_OSA_Error</i> | The mutex can not be unlocked or invalid parameter. |

Example:

```
osa_status_t status;
status = OSA_MutexUnlock(&myMutex);
switch (status)
{
    //...
}
```

47.3.8 osa_status_t OSA_MutexDestroy (mutex_t * pMutex)

Parameters

| | |
|---------------|-----------------------|
| <i>pMutex</i> | Pointer to the Mutex. |
|---------------|-----------------------|

Return values

| | |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | The mutex is successfully destroyed. |
| <i>kStatus_OSA_Error</i> | The mutex can not be destroyed. |

Example:

```
osa_status_t status;
status = OSA_MutexDestroy(&myMutex);
switch (status)
{
    //...
}
```

Function Documentation

47.3.9 osa_status_t OSA_EventCreate (event_t * *pEvent*, osa_event_clear_mode_t *clearMode*)

This function creates an event object and set its clear mode. If clear mode is kEventAutoClear, when a task gets the event flags, these flags will be cleared automatically. If clear mode is kEventManualClear, these flags must be cleared manually.

Parameters

| | |
|------------------|--|
| <i>pEvent</i> | Pointer to the event object to initialize. |
| <i>clearMode</i> | The event is auto-clear or manual-clear. |

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | The event object is successfully created. |
| <i>kStatus_OSA_Error</i> | The event object is not created. |

Example:

```
event_t myEvent;
OSA_EventCreate(&myEvent, kEventAutoClear);
```

47.3.10 osa_status_t OSA_EventWait (event_t * *pEvent*, event_flags_t *flagsToWait*, bool *waitForAll*, uint32_t *timeout*, event_flags_t * *setFlags*)

This function waits for a combination of flags to be set in an event object. Applications can wait for any/all bits to be set. Also this function could obtain the flags who wakeup the waiting task.

Parameters

| | |
|--------------------|---|
| <i>pEvent</i> | Pointer to the event. |
| <i>flagsToWait</i> | Flags that to wait. |
| <i>waitForAll</i> | Wait all flags or any flag to be set. |
| <i>timeout</i> | The maximum number of milliseconds to wait for the event. If the wait condition is not met, pass OSA_WAIT_FOREVER will wait indefinitely, pass 0 will return kStatus_OSA_Timeout immediately. |

| | |
|-----------------|--|
| <i>setFlags</i> | Flags that wakeup the waiting task are obtained by this parameter. |
|-----------------|--|

Return values

| | |
|----------------------------|--|
| <i>kStatus_OSA_Success</i> | The wait condition met and function returns successfully. |
| <i>kStatus_OSA_Timeout</i> | Has not met wait condition within timeout. |
| <i>kStatus_OSA_Error</i> | An incorrect parameter was passed. |
| <i>kStatus_OSA_Idle</i> | The wait condition is not met and 'timeout' is not exhausted, This is only for bare metal. |

Note

1. With bare metal, a event object can not be waited by more than one tasks at the same time.
 1. Please pay attention to the flags bit width, FreeRTOS uses the most significant 8 bits as control bits, so do not wait these bits while using FreeRTOS.

Example:

```
osa_status_t status;
event_flags_t setFlags;
status = OSA_EventWait(&myEvent, 0x01, true, 100, &setFlags);
switch (status)
{
    //...
}
```

47.3.11 **osa_status_t OSA_EventSet (event_t * *pEvent*, event_flags_t *flagsToSet*)**

Sets specified flags of an event object.

Parameters

| | |
|-------------------|-----------------------|
| <i>pEvent</i> | Pointer to the event. |
| <i>flagsToSet</i> | Flags to be set. |

Return values

| | |
|----------------------------|------------------------------------|
| <i>kStatus_OSA_Success</i> | The flags were successfully set. |
| <i>kStatus_OSA_Error</i> | An incorrect parameter was passed. |

Example:

```
osa_status_t status;
status = OSA_EventSet(&myEvent, 0x01);
switch (status)
{
    //...
}
```

Function Documentation

47.3.12 osa_status_t OSA_EventClear (*event_t * pEvent*, *event_flags_t flagsToClear*)

Clears specified flags of an event object.

Parameters

| | |
|---------------------|-----------------------|
| <i>pEvent</i> | Pointer to the event. |
| <i>flagsToClear</i> | Flags to be clear. |

Return values

| | |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | The flags were successfully cleared. |
| <i>kStatus_OSA_Error</i> | An incorrect parameter was passed. |

Example:

```
osa_status_t status;
status = OSA_EventClear(&myEvent, 0x01);
switch (status)
{
    //...
}
```

47.3.13 **event_flags_t OSA_EventGetFlags (event_t * *pEvent*)**

Gets the event flags status.

Parameters

| | |
|---------------|-----------------------|
| <i>pEvent</i> | Pointer to the event. |
|---------------|-----------------------|

Returns

event_flags_t Current event flags.

Example:

```
event_flags_t flags;
flags = OSA_EventGetFlags(&myEvent);
```

47.3.14 **osa_status_t OSA_EventDestroy (event_t * *pEvent*)**

Function Documentation

Parameters

| | |
|---------------|-----------------------|
| <i>pEvent</i> | Pointer to the event. |
|---------------|-----------------------|

Return values

| | |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | The event is successfully destroyed. |
| <i>kStatus_OSA_Error</i> | Event destruction failed. |

Example:

```
osa_status_t status;
status = OSA_EventDestroy(&myEvent);
switch (status)
{
    //...
}
```

47.3.15 **osa_status_t OSA_TaskCreate (task_t *task*, uint8_t * *name*, uint16_t *stackSize*, task_stack_t * *stackMem*, uint16_t *priority*, task_param_t *param*, bool *usesFloat*, task_handler_t * *handler*)**

This function is used to create task based on the resources defined by the macro OSA_TASK_DEFINE.

Parameters

| | |
|------------------|--|
| <i>task</i> | The task function entry. |
| <i>name</i> | The name of this task. |
| <i>stackSize</i> | The stack size in byte. |
| <i>stackMem</i> | Pointer to the stack. |
| <i>priority</i> | Initial priority of the task. |
| <i>param</i> | Pointer to be passed to the task when it is created. |
| <i>usesFloat</i> | This task will use float register or not. |
| <i>handler</i> | Pointer to the task handler. |

Return values

| | |
|----------------------------|-----------------------------------|
| <i>kStatus_OSA_Success</i> | The task is successfully created. |
|----------------------------|-----------------------------------|

| | |
|--------------------------|-------------------------------|
| <i>kStatus_OSA_Error</i> | The task can not be created.. |
|--------------------------|-------------------------------|

Example:

```
osa_status_t status;
OSA_TASK_DEFINE(task_func, stackSize);
status = OSA_TaskCreate(task_func,
                        "task_name",
                        stackSize,
                        task_func_stack,
                        prio,
                        param,
                        false,
                        &task_func_task_handler);

switch (status)
{
    //...
}
```

Note

Use the return value to check whether the task is created successfully. DO NOT check handler. For uC/OS-III, handler is not NULL even if the task creation has failed.

47.3.16 osa_status_t OSA_TaskDestroy (task_handler_t *handler*)

Parameters

| | |
|----------------|--|
| <i>handler</i> | The handler of the task to destroy. Returned by the OSA_TaskCreate function. |
|----------------|--|

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | The task was successfully destroyed. |
| <i>kStatus_OSA_Error</i> | Task destruction failed or invalid parameter. |

Example:

```
osa_status_t status;
status = OSA_TaskDestroy(myTaskHandler);
switch (status)
{
    //...
}
```

47.3.17 osa_status_t OSA_TaskYield (void)

When a task calls this function, it gives up the CPU and puts itself to the end of a task ready list.

Function Documentation

Return values

| | |
|----------------------------------|--------------------------------------|
| <code>kStatus_OSA_Success</code> | The function is called successfully. |
| <code>kStatus_OSA_Error</code> | Error occurs with this function. |

Example:

```
osa_status_t status;  
status = OSA_TaskYield();  
switch (status)  
{  
    //...  
}
```

47.3.18 `task_handler_t OSA_TaskGetHandler (void)`

Returns

Handler to current active task.

Example:

```
task_handler_t handler = OSA_TaskYield();
```

47.3.19 `uint16_t OSA_TaskGetPriority (task_handler_t handler)`

Parameters

| | |
|----------------------|---|
| <code>handler</code> | The handler of the task whose priority is received. |
|----------------------|---|

Returns

Task's priority.

Example:

```
uint16_t taskPrio = OSA_TaskGetPriority(taskHandler);
```

47.3.20 `osa_status_t OSA_TaskSetPriority (task_handler_t handler, uint16_t priority)`

Parameters

| | |
|-----------------|---|
| <i>handler</i> | The handler of the task whose priority is received. |
| <i>priority</i> | The priority to set. |

Return values

| | |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | Task's priority is set successfully. |
| <i>kStatus_OSA_Error</i> | Task's priority can not be set. |

Example:

```
osa_status_t status;
status = OSA_TaskSetPriority(taskHandler, newPrio);
switch (status)
{
    //...
}
```

47.3.21 msg_queue_handler_t OSA_MsgQCreate (msg_queue_t *queue, uint16_t message_number, uint16_t message_size)

This function initializes the message queue that was declared previously. This is an example demonstrating use:

```
msg_queue_handler_t handler;
MSG_QUEUE_DECLARE(my_message, msg_num, msg_size);
handler = OSA_MsgQCreate(my_message, msg_num, msg_size);
```

Parameters

| | |
|-----------------------|---|
| <i>queue</i> | The queue declared through the MSG_QUEUE_DECLARE macro. |
| <i>message_number</i> | The number of elements in the queue. |
| <i>message_size</i> | Size of every elements in words. |

Returns

Handler to access the queue for put and get operations. If message queue created failed, return 0.

47.3.22 osa_status_t OSA_MsgQPut (msg_queue_handler_t handler, void * pMessage)

This function puts a message to the end of the message queue. If the queue is full, this function returns the kStatus_OSA_Error;

Function Documentation

Parameters

| | |
|-----------------|--|
| <i>handler</i> | Queue handler returned by the OSA_MsgQCreate function. |
| <i>pMessage</i> | Pointer to the message to be put into the queue. |

Return values

| | |
|----------------------------|--|
| <i>kStatus_OSA_Success</i> | Message successfully put into the queue. |
| <i>kStatus_OSA_Error</i> | The queue was full or an invalid parameter was passed. |

Example:

```
osa_status_t status;
struct MESSAGE messageToPut = ...;
status = OSA_MsgQPut(queueHandler, &messageToPut);
switch (status)
{
    //...
}
```

47.3.23 **osa_status_t OSA_MsgQGet (msg_queue_handler_t *handler*, void * *pMessage*, uint32_t *timeout*)**

This function gets a message from the head of the message queue. If the queue is empty, timeout is used to wait.

Parameters

| | |
|-----------------|--|
| <i>handler</i> | Queue handler returned by the OSA_MsgQCreate function. |
| <i>pMessage</i> | Pointer to a memory to save the message. |
| <i>timeout</i> | The number of milliseconds to wait for a message. If the queue is empty, pass OSA_WAIT_FOREVER will wait indefinitely, pass 0 will return kStatus_OSA_Timeout immediately. |

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | Message successfully obtained from the queue. |
| <i>kStatus_OSA_Timeout</i> | The queue remains empty after timeout. |
| <i>kStatus_OSA_Error</i> | Invalid parameter. |

| | |
|-------------------------|---|
| <i>kStatus_OSA_Idle</i> | The queue is empty and 'timeout' is not exhausted, This is only for bare metal. |
|-------------------------|---|

Note

With bare metal, there should be only one process waiting on the queue.

Example:

```
osa_status_t status;
struct MESSAGE messageToGet;
status = OSA_MsgQGet(queueHandler, &messageToGet, 100);
switch (status)
{
    //...
}
```

47.3.24 osa_status_t OSA_MsgQDestroy (msg_queue_handler_t *handler*)

Parameters

| | |
|----------------|--|
| <i>handler</i> | Queue handler returned by the OSA_MsgQCreate function. |
|----------------|--|

Return values

| | |
|----------------------------|---------------------------------------|
| <i>kStatus_OSA_Success</i> | The queue was successfully destroyed. |
| <i>kStatus_OSA_Error</i> | Message queue destruction failed. |

Example:

```
osa_status_t status;
status = OSA_MsgQDestroy(queueHandler);
switch (status)
{
    //...
}
```

47.3.25 void* OSA_MemAlloc (size_t *size*)

Function Documentation

Parameters

| | |
|-------------|-----------------------------|
| <i>size</i> | Amount of bytes to reserve. |
|-------------|-----------------------------|

Returns

Pointer to the reserved memory. NULL if memory could not be allocated.

47.3.26 void* OSA_MemAllocZero (size_t *size*)

Parameters

| | |
|-------------|-----------------------------|
| <i>size</i> | Amount of bytes to reserve. |
|-------------|-----------------------------|

Returns

Pointer to the reserved memory. NULL if memory could not be allocated.

47.3.27 osa_status_t OSA_MemFree (void * *ptr*)

Parameters

| | |
|------------|---|
| <i>ptr</i> | Pointer to the start of the memory block previously reserved. |
|------------|---|

Return values

| | |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | Memory correctly freed. |
| <i>kStatus_OSA_Error</i> | Error occurs during free the memory. |

47.3.28 void OSA_TimeDelay (uint32_t *delay*)

Parameters

| | |
|--------------|-----------------------------------|
| <i>delay</i> | The time in milliseconds to wait. |
|--------------|-----------------------------------|

47.3.29 uint32_t OSA_TimeGetMsec (void)

Returns

Current time in milliseconds.

47.3.30 osa_status_t OSA_InstallIntHandler (int32_t *IRQNumber*, void(*)(void) *handler*)

Parameters

| | |
|------------------|-----------------------------------|
| <i>IRQNumber</i> | IRQ number of the interrupt. |
| <i>handler</i> | The interrupt handler to install. |

Return values

| | |
|----------------------------|------------------------------------|
| <i>kStatus_OSA_Success</i> | Handler is installed successfully. |
| <i>kStatus_OSA_Error</i> | Handler could not be installed. |

47.3.31 void OSA_EnterCritical (osa_critical_part_mode_t *mode*)

Parameters

| | |
|-------------|---|
| <i>mode</i> | Lock task scheduler or disable interrupt in critical part. Pass kCriticalLockSched to lock task scheduler, pass kCriticalDisableInt to disable interrupt. |
|-------------|---|

47.3.32 void OSA_ExitCritical (osa_critical_part_mode_t *mode*)

Parameters

| | |
|-------------|---|
| <i>mode</i> | Lock task scheduler or disable interrupt in critical part. Pass kCriticalLockSched to lock task scheduler, pass kCriticalDisableInt to disable interrupt. |
|-------------|---|

47.3.33 osa_status_t OSA_Init (void)

This function sets up the basic RTOS services. It should be called first in the main function.

Function Documentation

Return values

| | |
|----------------------------|---|
| <i>kStatus_OSA_Success</i> | RTOS services are initialized successfully. |
| <i>kStatus_OSA_Error</i> | Error occurs during initialization. |

47.3.34 osa_status_t OSA_Start (void)

This function starts the RTOS scheduler and may never return.

Return values

| | |
|----------------------------|----------------------------------|
| <i>kStatus_OSA_Success</i> | RTOS starts to run successfully. |
| <i>kStatus_OSA_Error</i> | Error occurs when start RTOS. |

47.4 Bare Metal Abstraction Layer

The Kinetis SDK provides the Bare Metal Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

47.4.1 Overview

Data Structures

- struct `semaphore_t`
Type for an semaphore. [More...](#)
- struct `mutex_t`
Type for a mutex. [More...](#)
- struct `event_t`
Type for an event object. [More...](#)
- struct `task_control_block_t`
Task control block for bare metal. [More...](#)
- struct `msg_queue_t`
Type for a message queue. [More...](#)

Macros

- #define `OSA_WAIT_FOREVER` 0xFFFFFFFFU
Constant to pass as timeout value in order to wait indefinitely.
- #define `TASK_MAX_NUM` 5
How many tasks can the bare metal support.
- #define `FSL_OSA_TIME_RANGE` 0xFFFFU
OSA's time range in millisecond, OSA time wraps if exceeds this value.

Typedefs

- typedef `uint32_t event_flags_t`
Type for an event flags group, bit 32 is reserved.
- typedef `void * task_param_t`
Type for task parameter.
- typedef `void(* task_t)(task_param_t param)`
Type for a task pointer.
- typedef `task_control_block_t * task_handler_t`
Type for a task handler, returned by the OSA_TaskCreate function.
- typedef `uint32_t task_stack_t`
Type for a task stack.
- typedef `msg_queue_t * msg_queue_handler_t`
Type for a message queue handler.

Bare Metal Abstraction Layer

Thread management

- void **OSA_PollAllOtherTasks** (void)
Calls all task functions one time except for the current task.
- #define **OSA_TASK_DEFINE**(task, stackSize)
Defines a task.

Message queues

- #define **MSG_QUEUE_DECLARE**(name, number, size)
This macro statically reserves the memory required for the queue.

47.4.1.1 Bare Metal Abstraction Layer

Overview

When RTOSes are not used, bare metal abstraction layer provides semaphore, mutex, event, message queue and so on. Because bare metal does not have a task scheduler, it is necessary to exercise caution while using bare metal abstraction layer.

Bare Metal's Task Management

By contrast to RTOSes, bare metal abstraction layer uses a poll mechanism to simulate a task. All task functions are linked into a list and called one by one. Therefore, bare metal task function should not contain an infinite loop. It must return at a proper time to let the other tasks run.

Bare metal task does not support priority, all tasks use the same priority. The macro **TASK_MAX_NUM** defines how many tasks applications could create. If it is set to 0, then applications could not use task APIs.

Bare Metal's Wait Functions

Bare metal wait functions, such as **OSA_SemaWait** and **OSA_EventWait**, return the **kStatus_OSA_Idle** if wait condition is not met and timeout has not occurred. Applications should catch this value and take proper actions. If the wait condition is set by the ISR, applications could wait in a loop:

```
void post_ISR(void)
{
    //...
    OSA_SemaPost(&my_sem);
    //...
}

void wait_task(task_param_t param)
{
```

```

//...
do
{
    status = OSA_SemaWait(&my_sem, 10);
} while(kStatus_OSA_Idle == status);

//...
}

```

In this example, if my_sem is not posted by post_ISR, but posted by a task post_task, then OSA_SemaWait in loop could never get my_sem within timeout, because post_task does not have chance to post my_sem. In this situation, applications could be implemented like this:

```

void post_task(task_param_t param)
{
    //...
    OSA_SemaPost(&my_sem);
    //...
}

void wait_task(task_param_t param)
{
    status = OSA_SemaWait(&my_sem, 10);

    switch (status)
    {
        case kStatus_OSA_Idle:
            return;
        case kStatus_OSA_Success:
            // ...
            break;
        case kStatus_OSA_Error:
            // ...
            break;
        case kStatus_OSA_Timeout:
            // ...
            break;
    }
    //...
}

```

Wait the semaphore at the start of the task, if kStatus_OSA_Idle is got, return and let other task run, then post_task has chance to post my_sem.

The other method is using function [OSA_PollAllOtherTasks\(\)](#). This function calls all other tasks one time, then post_task could post my_sem.

```

void post_task(task_param_t param)
{
    //...
    OSA_SemaPost(&my_sem);
    //...
}

void wait_task(task_param_t param)
{
    //...
    status = OSA_SemaWait(&my_sem, 10);

    while (kStatus_OSA_Idle == status)

```

Bare Metal Abstraction Layer

```
{  
    OSA_PollAllOtherTasks();  
    status = OSA_SemaWait(&my_sem, 10);  
}  
  
//...  
}
```

The limitation of this method is that only one task can use the [OSA_PollAllOtherTasks\(\)](#) function. If both task_A and task_B call this function, then the stack overflow may occur, because the call stack is like this: task_A -> OSA_PollAllOtherTasks -> task_B -> OSA_PollAllOtherTasks -> task_A -> ...

Bare Metal Time management

Bare metal OSA maintains a system time by the lowpower timer module. Applications can get system time in milliseconds using the function [OSA_TimeGetMsec\(\)](#). At the same time, all wait functions such as [OSA_SemaWait\(\)](#), [OSA_EventWait\(\)](#) depend on this system time. Lowpower timer module is set up in the function [OSA_Init\(\)](#). To use this time function, ensure that the [OSA_Init\(\)](#) function is called. Please note that lowpower timer provides only 16-bit time count, it wraps every 65536ms. So take care to use these three kinds of functions:

1. [OSA_TimeDelay\(\)](#) can not delay longer than 65536ms.
2. Wait functions, such as [OSA_SemaWait\(\)](#), can not set timeout longer than 65536ms, however, OS-A_WAIT_FOREVER is allowed.
3. [OSA_TimeGetMsec\(\)](#) wraps every 65536ms, if it does not meet the requirement, please implement use other timer module in application.

47.4.2 Data Structure Documentation

47.4.2.1 struct semaphore_t

Data Fields

- volatile bool [isWaiting](#)
Is any task waiting for a timeout on this object.
- volatile uint8_t [semCount](#)
The count value of the object.
- uint32_t [time_start](#)
The time to start timeout.
- uint32_t [timeout](#)
Timeout to wait in milliseconds.

47.4.2.2 struct mutex_t

Data Fields

- volatile bool [isWaiting](#)

- volatile bool **isLocked**
Is any task waiting for a timeout on this mutex.
- uint32_t **time_start**
The time to start timeout.
- uint32_t **timeout**
Timeout to wait in milliseconds.
- LWSEM_STRUCT **sema**
The lwsem structure.
- **_task_id owner**
Task who locks this mutex.

47.4.2.2.0.52 Field Documentation

47.4.2.2.0.52.1 LWSEM_STRUCT mutex_t::sema

47.4.2.2.0.52.2 _task_id mutex_t::owner

47.4.2.3 struct event_t

Data Fields

- volatile bool **isWaiting**
Is any task waiting for a timeout on this event.
- uint32_t **time_start**
The time to start timeout.
- uint32_t **timeout**
Timeout to wait in milliseconds.
- volatile event_flags_t **flags**
The flags status.
- osa_event_clear_mode_t **clearMode**
Auto clear or manual clear.

47.4.2.4 struct task_control_block_t

Data Fields

- task_t **p_func**
Task's entry.
- task_param_t **param**
Task's parameter.
- struct TaskControlBlock * **next**
Pointer to next task control block.
- struct TaskControlBlock * **prev**
Pointer to previous task control block.

47.4.2.5 struct msg_queue_t

Data Fields

- `uint32_t * queueMem`
Points to the queue memory.
- `uint16_t number`
The number of messages in the queue.
- `uint16_t size`
The size in words of each message.
- `uint16_t head`
Index of the next message to be read.
- `uint16_t tail`
Index of the next place to write to.
- `semaphore_t queueSem`
Semaphore wakeup tasks waiting for msg.
- volatile bool `isEmpty`
Whether queue is empty.

47.4.3 Macro Definition Documentation

47.4.3.1 #define OSA_WAIT_FOREVER 0xFFFFFFFFU

47.4.3.2 #define TASK_MAX_NUM 5

47.4.3.3 #define FSL_OSA_TIME_RANGE 0xFFFFU

47.4.3.4 #define OSA_TASK_DEFINE(task, stackSize)

Value:

```
task_stack_t* task##_stack = NULL; \
task_handler_t task##_task_handler
```

This macro defines resources for a task statically. Then, the OSA_TaskCreate creates the task based-on these resources.

Parameters

| | |
|------------------------|--|
| <code>task</code> | The task function. |
| <code>stackSize</code> | The stack size this task needs in bytes. |

47.4.3.5 #define MSG_QUEUE_DECLARE(name, number, size)

Value:

```

uint32_t queueMem_##name[number * size]; \
entity_##name = { \
    .queueMem = queueMem_##name \
}; \
*name = &(entity_##name)

```

Parameters

| | |
|---------------|-----------------------------------|
| <i>name</i> | Identifier for the memory region. |
| <i>number</i> | Number of elements in the queue. |
| <i>size</i> | Size of every element in words. |

47.4.4 Function Documentation

47.4.4.1 void OSA_PollAllOtherTasks(void)

This function calls all other task functions one time. If current task is waiting for an event triggered by other tasks, this function could be used to trigger the event.

Note

There should be only one task calls this function, if more than one task call this function, stack overflow may occurs. Be careful to use this function.

MQX Abstraction Layer

47.5 MQX Abstraction Layer

The Kinetis SDK provides the MQX Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

47.5.1 Overview

Data Structures

- struct `mutex_t`
Type for a mutex. [More...](#)

Macros

- #define `OSA_WAIT_FOREVER` 0xFFFFFFFFFU
Constant to pass as timeout value in order to wait indefinitely.
- #define `FSL_OSA_TIME_RANGE` 0xFFFFFFFFFU
OSA's time range in millisecond, OSA time wraps if exceeds this value.
- #define `MQX_MAIN_TASK_PRIORITY` (7+16)
The priority of MQX's Main_Task.

Typedefs

- typedef LWSEM_STRUCT `semaphore_t`
Type for MQX mutex.
- typedef LWEVENT_STRUCT `event_t`
Type for an event group object.
- typedef _mqx_uint `event_flags_t`
Type for an event flags group, bit 32 is reserved.
- typedef TASK_FPTR `task_t`
Type for a task pointer.
- typedef _task_id `task_handler_t`
Type for a task handler, returned by the OSA_TaskCreate function.
- typedef uint32_t `task_param_t`
Type for a task handler, returned by the task_create function.
- typedef uint32_t `task_stack_t`
Type for a task stack.
- typedef _mqx_max_type `msg_queue_t`
Type for a message queue.
- typedef void * `msg_queue_handler_t`
Type for a message queue handler.

Thread management

- #define `OSA_TASK_DEFINE`(task, stackSize)

- `#define PRIORITY_OSA_TORTOS(osa_prio) ((osa_prio)+7U)`
To provide unified task priority for upper layer, OSA layer makes conversion.
- `#define PRIORITY_RTOS_TOOSA(rtos_prio) ((rtos_prio)-7U)`

Message queues

- `void * OSA_MemoryAllocateAlign (size_t size, size_t align)`
Allocates the block aligned at a specific boundary.
- `#define SIZE_IN_MMT_UNITS(size) ((size + sizeof(_mqx_max_type) - 1) / sizeof(_mqx_max_type))`
- `#define MSG_QUEUE_DECLARE(name, number, size) _mqx_max_type name[SIZE_IN_MMT_UNITS(sizeof(LWMSGQ_STRUCT)) + SIZE_IN_MMT_UNITS(size * 4) * number]`
This macro statically reserves the memory required for the queue.

47.5.2 Data Structure Documentation

47.5.2.1 struct mutex_t

Data Fields

- `volatile bool isWaiting`
Is any task waiting for a timeout on this mutex.
- `volatile bool isLocked`
Is the object locked or not.
- `uint32_t time_start`
The time to start timeout.
- `uint32_t timeout`
Timeout to wait in milliseconds.
- `LWSEM_STRUCT sema`
The lwsem structure.
- `_task_id owner`
Task who locks this mutex.

MQX Abstraction Layer

47.5.2.1.0.53 Field Documentation

47.5.2.1.0.53.1 LWSEM_STRUCT mutex_t::sema

47.5.2.1.0.53.2 _task_id mutex_t::owner

47.5.3 Macro Definition Documentation

47.5.3.1 #define OSA_WAIT_FOREVER 0xFFFFFFFFFU

47.5.3.2 #define FSL_OSA_TIME_RANGE 0xFFFFFFFFFU

47.5.3.3 #define MQX_MAIN_TASK_PRIORITY (7+16)

47.5.3.4 #define OSA_TASK_DEFINE(task, stackSize)

Value:

```
task_stack_t task##_stack[MQX_REQUIRED_STACK_SIZE((stackSize)/sizeof(  
    task_stack_t))+1]; \  
task_handler_t task##_task_handler;
```

This macro defines resources for a task statically. Then, the OSA_TaskCreate creates the task based-on these resources.

Parameters

| | |
|------------------|--|
| <i>task</i> | The task function. |
| <i>stackSize</i> | The stack size this task needs in bytes. |

47.5.3.5 #define PRIORITY_OSA_TO_RTOS(*osa_prio*) ((*osa_prio*)+7U)

MQX's highest 7 priorities are special priorities.

47.5.3.6 #define MSG_QUEUE_DECLARE(*name*, *number*, *size*) _mqx-
_max_type name[SIZE_IN_MMT_UNITS(sizeof(LWMSGQ_STRUCT)) +
SIZE_IN_MMT_UNITS(*size* * 4) * *number*]

Parameters

| | |
|---------------|-----------------------------------|
| <i>name</i> | Identifier for the memory region. |
| <i>number</i> | Number of elements in the queue. |
| <i>size</i> | Size of element in 4B units. |

47.5.4 TYPEDOC Documentation

47.5.4.1 **typedef LWSEM_STRUCT semaphore_t**

Type for a semaphore.

47.5.4.2 **typedef LWEVENT_STRUCT event_t**

47.5.4.3 **typedef _mqx_uint event_flags_t**

47.5.4.4 **typedef TASK_FPTR task_t**

47.5.4.5 **typedef _task_id task_handler_t**

47.5.4.6 **typedef uint32_t task_param_t**

47.5.4.7 **typedef uint32_t task_stack_t**

47.5.4.8 **typedef _mqx_max_type msg_queue_t**

47.5.4.9 **typedef void* msg_queue_handler_t**

47.6 μC/OS-II Abstraction Layer

The Kinetis SDK provides the μC/OS-II Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

47.6.1 Overview

Data Structures

- struct [event_ucosii](#)
Type for an event group object in μCOS-II. [More...](#)
- struct [msgq_ucosii](#)
Type for message queue in μCOS-II. [More...](#)

Macros

- #define [OSA_WAIT_FOREVER](#) 0xFFFFFFFFFU
Constant to pass as timeout value to wait indefinitely.
- #define [FSL_OSA_TIME_RANGE](#) 0xFFFFFFFFFU
OSA's time range in millisecond, OSA time wraps if exceeds this value.

Typedefs

- typedef OS_FLAGS [event_flags_t](#)
Type for an event flags group.
- typedef OS_EVENT * [semaphore_t](#)
Type for an semaphore.
- typedef OS_EVENT * [mutex_t](#)
Type for a mutex.
- typedef [event_ucosii event_t](#)
Type for an event group object.
- typedef [msgq_ucosii msg_queue_t](#)
Type for a message queue.
- typedef [msgq_ucosii * msg_queue_handler_t](#)
Type for a message queue handler.
- typedef OS_TCB * [task_handler_t](#)
Type for a task handler, returned by the OSA_TaskCreate function.
- typedef OS_STK [task_stack_t](#)
Type for a task stack.
- typedef void * [task_param_t](#)
Type for task parameter.
- typedef void(* [task_t](#))([task_param_t](#) param)
Type for a task pointer.

Thread management

- #define **OSA_TASK_DEFINE**(task, stackSize)
Defines a task.
- #define **PRIORITY_OSA_TORTOS**(osa_prio) ((osa_prio)+4U)
To provide unified task priority for upper layer; OSA layer makes conversion.
- #define **PRIORITY_RTOS_TOOSA**(rtos_prio) ((rtos_prio)-4U)

Message queues

- #define **MSG_QUEUE_DECLARE**(name, number, size)
This macro statically reserves the memory required for the queue.

47.6.2 Data Structure Documentation

47.6.2.1 struct event_ucosii

Data Fields

- OS_FLAG_GRP * pGroup
Pointer to μCOS-II's event entity.
- osa_event_clear_mode_t clearMode
Auto clear or manual clear.

47.6.2.2 struct msgq_ucosii

Data Fields

- OS_EVENT * pQueue
Pointer to the queue.
- void ** msgTbl
Pointer to the array that saves the pointers to messages.
- OS_MEM * pMem
Pointer to memory where save the messages.
- void * msgs
Memory to save the messages.
- uint16_t size
Size of the message in words.

47.6.3 Macro Definition Documentation

47.6.3.1 #define OSA_WAIT_FOREVER 0xFFFFFFFFU

47.6.3.2 #define FSL_OSA_TIME_RANGE 0xFFFFFFFFU

47.6.3.3 #define OSA_TASK_DEFINE(*task*, *stackSize*)

Value:

```
task_stack_t task##_stack[(stackSize)/sizeof(task_stack_t)];  
\\  
task_handler_t task##_task_handler
```

This macro defines resources for a task statically. Then the OSA_TaskCreate creates the task based on these resources.

Parameters

| | |
|------------------|--|
| <i>task</i> | The task function. |
| <i>stackSize</i> | The stack size this task needs in bytes. |

47.6.3.4 #define MSG_QUEUE_DECLARE(*name*, *number*, *size*)

Value:

```
void* msgTbl_##name[number];  
uint32_t msgs_##name[number*size];  
msg_queue_t memory_##name = {  
    .msgTbl = msgTbl_##name,  
    .msgs = msgs_##name  
};  
msg_queue_t *name = &(memory_##name)
```

Parameters

| | |
|---------------|-----------------------------------|
| <i>name</i> | Identifier for the memory region. |
| <i>number</i> | Number of elements in the queue. |
| <i>size</i> | Size of every elements in words. |

47.7 μC/OS-III Abstraction Layer

The Kinetis SDK provides the μC/OS-III Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

47.7.1 Overview

Data Structures

- struct [event_ucosiii](#)
Type for an event group object in μCOS-III. [More...](#)
- struct [msgq_struct_ucosiii](#)
Type for message queue in μCOS-III. [More...](#)

Macros

- #define [OSA_WAIT_FOREVER](#) 0xFFFFFFFFFU
Constant to pass as timeout value in order to wait indefinitely.
- #define [FSL_OSA_TIME_RANGE](#) 0xFFFFFFFFFU
OSA's time range in millisecond, OSA time wraps if exceeds this value.

Typedefs

- typedef OS_TCB * [task_handler_t](#)
Type for a task handler, returned by the OSA_TaskCreate function.
- typedef CPU_STK [task_stack_t](#)
Type for a task stack.
- typedef void * [task_param_t](#)
Type for task parameter.
- typedef void(* [task_t](#))([task_param_t](#) param)
Type for a task pointer.
- typedef OS_SEM [semaphore_t](#)
Type for a semaphore.
- typedef OS_MUTEX [mutex_t](#)
Type for a mutex.
- typedef OS_FLAGS [event_flags_t](#)
Type for an event flags group.
- typedef [event_ucosiii](#) [event_t](#)
Type for an event object.
- typedef [msgq_struct_ucosiii](#) [msg_queue_t](#)
Type for a message queue.
- typedef [msg_queue_t](#) * [msg_queue_handler_t](#)
Type for a message queue handler.

Thread management

- #define **OSA_TASK_DEFINE**(task, stackSize)
Defines a task.
- #define **PRIORITY_OSA_TO_RTOS**(osa_prio) ((osa_prio)+1U)
To provide unified task priority for upper layer; OSA layer makes conversion.
- #define **PRIORITY_RTOS_TO_OSA**(rtos_prio) ((rtos_prio)-1U)

Message queues

- #define **MSG_QUEUE_DECLARE**(name, number, size)
This macro statically reserves the memory required for the queue.

47.7.2 Data Structure Documentation

47.7.2.1 struct event_ucosiii

Data Fields

- OS_FLAG_GRP **group**
μCOS-III's event entity.
- **osa_event_clear_mode_t clearMode**
Auto clear or manual clear.

47.7.2.2 struct msgq_struct_ucosiii

Data Fields

- OS_Q **queue**
the message queue's control block
- OS_MEM **mem**
control block for the memory where save the messages
- void * **msgs**
pointer to the memory where save the messages
- uint16_t **size**
size of the message in words

47.7.3 Macro Definition Documentation

47.7.3.1 #define OSA_WAIT_FOREVER 0xFFFFFFFFFU

47.7.3.2 #define FSL_OSA_TIME_RANGE 0xFFFFFFFFFU

47.7.3.3 #define OSA_TASK_DEFINE(task, stackSize)

Value:

```
OS_TCB TCB_##task;
  task_stack_t task##_stack[(stackSize)/sizeof(task_stack_t)];
  \
  task_handler_t task##_task_handler = &(TCB_##task)
```

This macro defines resources for a task statically. Then, the OSA_TaskCreate creates the task based on these resources.

Parameters

| | |
|------------------|--|
| <i>task</i> | The task function. |
| <i>stackSize</i> | The stack size this task needs in bytes. |

47.7.3.4 #define PRIORITY_OSA_TO_RTOS(osa_prio) ((osa_prio)+1U)

uC/OS-III's tick task should have a high priority, so we set tick task to priority 0, applications use other priorities.

47.7.3.5 #define MSG_QUEUE_DECLARE(name, number, size)

Value:

```
uint32_t msgs_##name[number*size];
msg_queue_t memory_##name = {
    .msgs = msgs_##name
};
msg_queue_t *name = &(memory_##name)
```

Parameters

| | |
|---------------|-----------------------------------|
| <i>name</i> | Identifier for the memory region. |
| <i>number</i> | Number of elements in the queue. |
| <i>size</i> | Size of every elements in words. |

47.7.4 Typedef Documentation

47.7.4.1 **typedef OS_TCB* task_handler_t**

47.7.4.2 **typedef CPU_STK task_stack_t**

47.7.4.3 **typedef OS_SEM semaphore_t**

47.7.4.4 **typedef OS_MUTEX mutex_t**

47.7.4.5 **typedef OS_FLAGS event_flags_t**

47.7.4.6 **typedef event_ucosiii event_t**

47.7.4.7 **typedef msgq_struct_ucosiii msg_queue_t**

47.7.4.8 **typedef msg_queue_t* msg_queue_handler_t**

47.8 FreeRTOS Abstraction Layer

The Kinetis SDK provides the FreeRTOS Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

47.8.1 Overview

Data Structures

- struct `event_freertos`
Type for an event group object in FreeRTOS. [More...](#)

Macros

- #define `OSA_WAIT_FOREVER` 0xFFFFFFFFFU
Constant to pass as timeout value in order to wait indefinitely.
- #define `FSL_OSA_TIME_RANGE` 0xFFFFFFFFFU
OSA's time range in millisecond, OSA time wraps if exceeds this value.

Typedefs

- typedef TaskHandle_t `task_handler_t`
Type for a task handler, returned by the `OSA_TaskCreate` function.
- typedef portSTACK_TYPE `task_stack_t`
Type for a task stack.
- typedef void * `task_param_t`
Type for task parameter.
- typedef pdTASK_CODE `task_t`
Type for a task function.
- typedef xSemaphoreHandle `mutex_t`
Type for a mutex.
- typedef xSemaphoreHandle `semaphore_t`
Type for a semaphore.
- typedef EventBits_t `event_flags_t`
Type for an event flags object.
- typedef `event_freertos event_t`
Type for an event group object.
- typedef xQueueHandle `msg_queue_t`
Type for a message queue declaration and creation.
- typedef xQueueHandle `msg_queue_handler_t`
Type for a message queue handler.

Thread management

- #define `OSA_TASK_DEFINE`(task, stackSize)

FreeRTOS Abstraction Layer

- `#define PRIORITY_OSA_TO_RTOS(osa_prio) (configMAX_PRIORITIES - (osa_prio) -2)`
To provide unified task priority for upper layer, OSA layer makes conversion.
- `#define PRIORITY_RTOS_TO_OSA(rtos_prio) (configMAX_PRIORITIES - (rtos_prio) -2)`

Message queues

- `#define MSG_QUEUE_DECLARE(name, number, size) msg_queue_t *name = NULL`
This macro statically reserves the memory required for the queue.

47.8.2 Data Structure Documentation

47.8.2.1 struct event_freertos

Data Fields

- `EventGroupHandle_t eventHandler`
FreeRTOS event handler.
- `osa_event_clear_mode_t clearMode`
Auto clear or manual clear.

47.8.3 Macro Definition Documentation

47.8.3.1 #define OSA_WAIT_FOREVER 0xFFFFFFFFU

47.8.3.2 #define FSL_OSA_TIME_RANGE 0xFFFFFFFFU

47.8.3.3 #define OSA_TASK_DEFINE(task, stackSize)

Value:

```
task_stack_t* task##_stack = NULL; \
    task_handler_t task##_task_handler
```

Parameters

| | |
|------------------------|--|
| <code>task</code> | The task function. |
| <code>stackSize</code> | Number of elements in the stack for this task. |

47.8.3.4 #define MSG_QUEUE_DECLARE(name, number, size) msg_queue_t *name = NULL

Parameters

| | |
|---------------|-----------------------------------|
| <i>name</i> | Identifier for the memory region. |
| <i>number</i> | Number of elements in the queue. |
| <i>size</i> | Size of every elements in words. |

47.8.4 Typedef Documentation

47.8.4.1 **typedef TaskHandle_t task_handler_t**

47.8.4.2 **typedef portSTACK_TYPE task_stack_t**

47.8.4.3 **typedef pdTASK_CODE task_t**

47.8.4.4 **typedef xSemaphoreHandle mutex_t**

47.8.4.5 **typedef xSemaphoreHandle semaphore_t**

47.8.4.6 **typedef EventBits_t event_flags_t**

47.8.4.7 **typedef xQueueHandle msg_queue_t**

How to Reach Us:

Home Page:

freescale.com

Web Support:

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale 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.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address:
freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, Kinetis, Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. 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.

© 2014 Freescale Semiconductor, Inc.

